Zajavka - Wstęp do programowania

Spis treści

Kilka słów od autorów

Cześć! Stworzyliśmy tę książkę, żeby pomóc Ci wejść w arcyciekawy świat programowania i pokazać Ci jak działają podstawy podstaw tego świata.

Naszym celem jest przedstawienie Ci podstawowych pojęć i mechanizmów, z jakimi zetkniesz się na swojej programistycznej ścieżce. Oczywiście robimy to po to, żeby zafascynować Cię tym światem, tak samo, jak my jesteśmy nim zafascynowani. Jednak chcemy zaznaczyć, że książka ta nie będzie wyjaśniała Ci wielu pojęć dogłębnie, mnóstwo z nich będzie poruszanych i przerabianych w trakcie faktycznego kursu Zajavka. W ramach tej książki chcemy dać Ci wprowadzanie do świata tworzenia oprogramowania i przedstawić (oraz w bardzo podstawowym stopniu wyjaśnić) pojęcia, z jakimi zetkniesz się na swojej programistycznej drodze.

W tym miejscu chcemy zaznaczyć bardzo ważną kwestię. Jeżeli uważasz, że masz już podstawy do wejścia w świat programowania, zacznij od zapoznania się ze spisem treści. Czujesz, że umiesz wypowiedzieć się na każde z postawionych w tej książce pytań? To znaczy, że możesz przechodzić bezpośrednio do naszego kursu. Zachęcamy jednak każdego do zapoznania się z tą książką! ☺

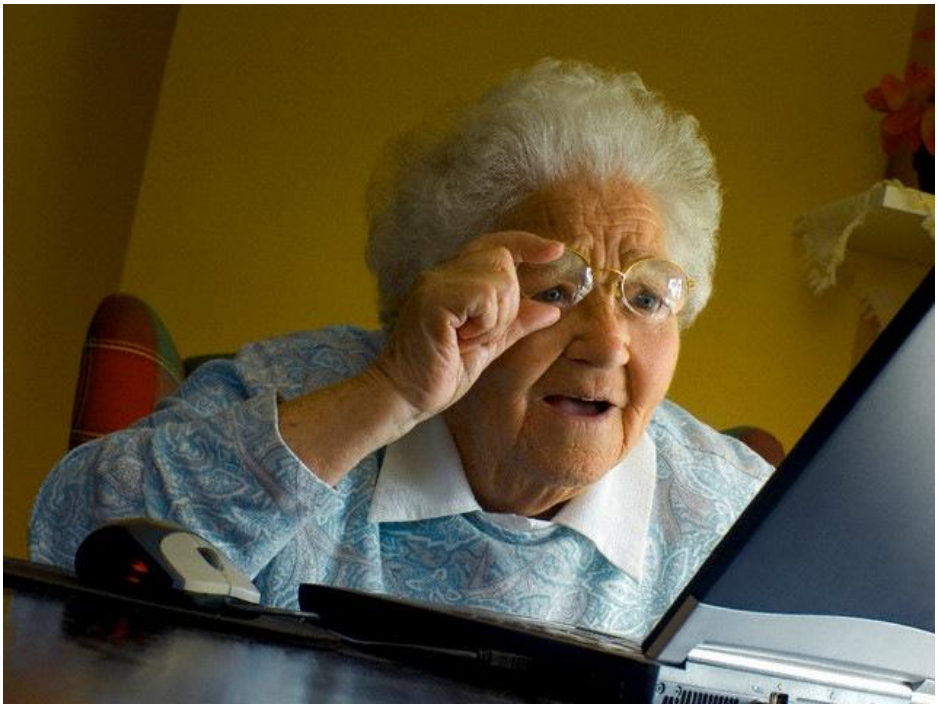
Książka ta jest zorganizowana w formie pytań i odpowiedzi. Przygotowaliśmy to w ten sposób, żeby później było Ci łatwo wrócić do podstawowych pojęć.

Chcemy w tym miejscu wyraźnie zaznaczyć, że w ramach tej książki będą pojawiały się fragmenty kodu, które nie będą na tym etapie dla Ciebie zrozumiałe w 100% i to jest normalne. Szczegółowe wyjaśnienia poruszanych w tej książce zagadnień będą w kursie Zajavka. Ta książka ma Ci tylko zajawić temat i dać ogólne poczucie, że wiesz, w którym kościele dzwonią dzwony.

Dlatego nie oczekuj od siebie, że po przeczytaniu tej książki będziesz wszystko pamiętać i rozumieć. To przyjdzie z czasem.

Kawa zrobiona? No to w drogę! ☺

O co w tym wszystkim chodzi?



Obraz 1. Źródło: https://imgflip.com/memegenerator/Grandma-Finds-The-Internet

Czym jest komputer?

Jak mawiała mama: *"Ledwo oczy otworzy, już do kąkutera leci!"*. Z pytaniem o komputer jest jak z pytaniem o czas, póki się nikt nas nie zapyta, wiemy doskonale, ale gdy mamy wytłumaczyć, pojawia się już problem.

Komputer to kompleksowe elektroniczne urządzenie, które wykonuje przetwarzanie informacji oraz prowadzi obliczenia na podstawie zdefiniowanych instrukcji. Składa się z różnych komponentów, takich jak centralna jednostka obliczeniowa (**CPU**), pamięć operacyjna (**RAM**), pamięć masowa (np. dysk twardy), klawiatura, monitor, mysz i inne urządzenia wejścia/wyjścia.

Podstawowym zadaniem komputera jest przetwarzanie danych, które mogą być reprezentowane w postaci binarnej (zer i jedynek). Komputer wykonuje operacje logiczne i arytmetyczne na tych danych, zgodnie z programem, który jest sekwencją instrukcji. Programy są tworzone przez programistów i

określają, jakie operacje mają zostać wykonane na danych.

Komputery mają różnorodne zastosowania w dziedzinach takich jak nauka, biznes, przemysł, medycyna, rozrywka i wiele innych. Mogą służyć do przetwarzania tekstu, tworzenia grafiki, analizy danych, symulacji, komunikacji, a nawet do sterowania innymi urządzeniami.

Postęp technologiczny sprawił, że komputery stały się coraz bardziej zaawansowane, szybkie, wydajne i powszechne. Obecnie mamy do czynienia z różnymi rodzajami komputerów, takimi jak komputery osobiste (PC), laptopy, tablety, smartfony, a także superkomputery, które są w stanie przetwarzać ogromne ilości danych w bardzo krótkim czasie.

Można zatem powiedzieć, że komputer to taki magiczny gadżet, który mieszka w twoim pokoju i wykonuje zadania, o których nawet nie miałeś/-aś pojęcia. To tak jakbyś miał/-a małego pomocnika, który jest trochę dziwaczny, ale zawsze gotowy do działania. Czyli coś jak Stefan, kolega z pracy.

Inne często spotykane określenie na komputer to: "*szybki idiota*" (Stefan, czy Ty przypadkiem na pewno nie jesteś komputerem?). Zrobi wszystko, o co go poprosisz i zrobi to bardzo szybko ☺.

Jak działa komputer?

Obraz 2. Źródło: Hacker Stock Photos - Trying to Hack and Hammer His Way In

Co to za pytanie? Stukasz w klawisze kąkutera a kąkuter gra i buczy! A tak na poważnie komputer działa na podstawie złożonej architektury i procesów, które pozwalają mu wykonywać różnorodne zadania. Ogólnie rzecz biorąc, działanie komputera można podzielić na cztery podstawowe etapy:

- Pobieranie danych;
- Przetwarzanie;
- Przechowywanie;
- Wyprowadzanie wyników.

Pierwszym krokiem jest wprowadzenie danych do komputera za pomocą urządzeń wejścia, takich jak klawiatura, mysz, mikrofon czy skaner. Dane te są następnie przechowywane w pamięci operacyjnej (RAM), gdzie są dostępne dla procesora.

Procesor, będący *"mózgiem"* komputera, wykonuje kolejne instrukcje, które są zawarte w programie. Instrukcje te są odczytywane z pamięci i przetwarzane wewnątrz procesora. Przetwarzanie obejmuje operacje logiczne (np. porównywanie danych) oraz operacje arytmetyczne (np. dodawanie, mnożenie). Procesor może również korzystać z pamięci podręcznej (cache) w celu przyspieszenia dostępu do danych.

W trakcie przetwarzania komputer może korzystać z danych przechowywanych w pamięci masowej, takiej jak dysk twardy. Pamięć masowa służy do długotrwałego przechowywania danych, nawet po wyłączeniu komputera. Na dysku twardym znajdują się pliki, programy oraz system operacyjny.

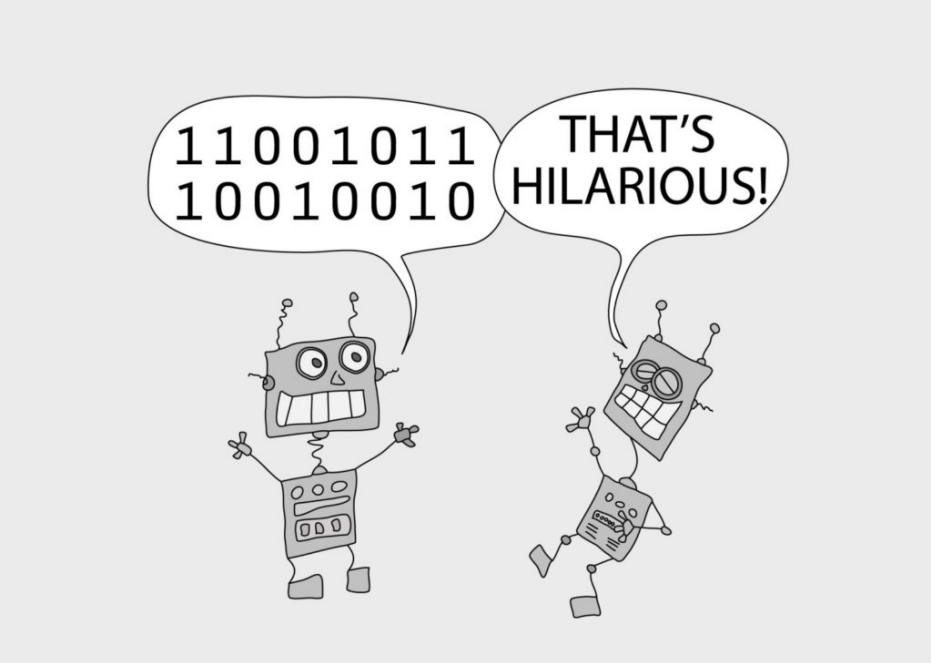
Po przetworzeniu danych wyniki są przechowywane lub przekazywane do odpowiednich urządzeń wyjściowych, takich jak monitor, drukarka, głośniki czy dysk zewnętrzny. Wyjścia te umożliwiają użytkownikowi odbiór i wykorzystanie wyników przetwarzania.

Komputer operuje na zasadzie cyklicznego powtarzania tych czterech podstawowych etapów: pobieranie danych, przetwarzanie, przechowywanie i wyprowadzanie wyników. To pozwala na wykonywanie skomplikowanych obliczeń, manipulowanie danymi, wykonywanie operacji na wielu programach jednocześnie i obsługę różnorodnych zadań.

Oczywiście, w tle działa również system operacyjny (wrócimy jeszcze do tego), który zarządza zasobami komputera, kontroluje urządzenia, umożliwia interakcję z użytkownikiem i zapewnia bezpieczeństwo działania. To całe towarzystwo składa się na to, jak komputer działa i jak może spełniać nasze potrzeby w cyfrowym świecie. Albo na to, jak zamula i się zawiesza. ☺

Jeżeli interesuje Cię jak dokładnie działa komputer *"pod spodem"*, polecamy zapoznać się z tym filmem: link do YouTube (Jak działa komputer?).

Dlaczego mówi się, że komputer rozumie tylko 0 i 1?



Obraz 3. Źródło: 10 Hilarious Binary Jokes That Will Make You Laugh | Convert Binary

Niby taki mądry a tylko do jednego liczyć potrafi! Mówi się, że komputer rozumie tylko 0 i 1, ponieważ jego podstawową jednostką informacji jest **bit**, który może przyjąć jedną z dwóch wartości: **zero lub**

jeden. Jest to związane z zastosowaniem systemu binarnego w komputerach.

System binarny to system liczbowy, który opiera się na dwóch cyfrach: zero i jeden. Komputery wykorzystują tę dwucyfrową reprezentację, ponieważ jest ona idealna do przedstawiania informacji za pomocą stanów logicznych. Zero i jeden odpowiadają dwóm stanom fizycznym, takim jak brak napięcia elektrycznego (0) i obecność napięcia elektrycznego (1) w komponentach elektronicznych. Tylko tyle i aż tyle.

Przez zastosowanie reprezentacji binarnej, komputery mogą przetwarzać i przechowywać dane w postaci sekwencji zer i jedynek. Te sekwencje, zwane kodami binarnymi, są interpretowane przez komputer jako instrukcje, symbole, liczby i inne formy informacji.

Dlatego mówimy, że komputer *"rozumie"* tylko 0 i 1, ponieważ wszystkie informacje, które przetwarza, są kodowane i przetwarzane w postaci binarnej. Chociaż dla ludzi czytanie i interpretowanie takich sekwencji może być trudne, dla komputera jest to naturalny sposób reprezentacji i manipulacji danych.

Przez przetwarzanie i kombinację zer i jedynek w odpowiednich wzorcach, komputer może wykonywać operacje logiczne, arytmetyczne, przechowywać informacje, generować grafikę, wykonywać obliczenia naukowe i wiele innych zadań. Ostatecznie, mówienie, że komputer *"rozumie"* tylko 0 i 1 odnosi się do sposobu, w jaki reprezentuje i operuje na danych w świecie cyfrowym.

System dziesiętny to najbardziej powszechny system liczbowy używany przez ludzi. Opiera się na dziesięciu cyfrach: *0, 1, 2, 3, 4, 5, 6, 7, 8 i 9*. W tym systemie każda kolejna cyfra ma wartość dziesięciokrotnie większą od poprzedniej.

W systemie dziesiętnym liczby są zapisywane za pomocą pozycyjnego systemu liczbowego, gdzie każda cyfra zajmuje określoną pozycję, a jej wartość jest mnożona przez odpowiednią potęgę liczby 10, zależnie od pozycji. Na przykład liczba 452 w systemie dziesiętnym oznacza, że mamy 4 setki, 5 dziesiątek i 2 jedności. Możemy to zapisać w taki sposób: $452 = 4 \times 10^2 + 5 \times 10^1 + 2 \times 10^0$.

System dziesiętny jest szeroko stosowany w codziennym życiu, ponieważ jest naturalny dla naszego sposobu liczenia (ludzie mają przecież 10 palców!). Ułatwia nam rozumienie wartości liczbowych i wykonywanie działań arytmetycznych, takich jak dodawanie, odejmowanie, mnożenie i dzielenie.

Istnieją również inne systemy liczbowe, takie jak binarny (wspomniany wcześniej, oparty na liczbie 2), ósemkowy (oparty na liczbie 8) czy szesnastkowy (oparty na liczbie 16), jednak to system dziesiętny stał się najbardziej rozpowszechnionym na świecie.

Więc nawet jeśli potrafisz liczyć tylko do dziesięciu (i to na palcach) to i tak potrafisz rozróżnić więcej cyfr niż komputer. 😉

Jak zbudowany jest komputer?

Obraz 4. Źródło: PC Builder Guide from HP

Stare informatyczne przysłowie mówi: *"Jak sobie komputer zbudujesz, tak Ci się będzie wywalał system"*. Komputer składa się z różnych komponentów, które współpracują ze sobą, tworząc kompleksowe urządzenie do przetwarzania informacji. Poniżej wymienimy podstawowe elementy budulcowe komputera:

1. **Centralna jednostka obliczeniowa (CPU):** Jest to *"mózg"* komputera, odpowiedzialny za wykonywanie instrukcji i obliczenia. Składa się z dwóch głównych części: jednostki kontrolnej, która zarządza operacjami i koordynuje działanie komputera, oraz jednostki arytmetyczno-logicznej (ALU), która wykonuje operacje matematyczne i logiczne.
2. **Pamięć operacyjna (RAM):** Jest to miejsce, gdzie komputer przechowuje bieżące dane i programy, z którymi aktualnie pracuje. Pamięć RAM jest ulotna, co oznacza, że dane są przechowywane tylko podczas pracy komputera i są tracone po wyłączeniu zasilania.
3. **Pamięć masowa:** Składa się z różnych urządzeń, takich jak dysk twardy, dysk SSD (nośnik półprzewodnikowy), napęd optyczny (np. CD/DVD) itp. Pamięć masowa jest używana do długotrwałego przechowywania danych, takich jak system operacyjny, pliki programów, dokumenty, multimedia itp.
4. **Klawiatura i mysz:** Służą do wprowadzania danych i sterowania komputerem. Klawiatura pozwala na wprowadzanie tekstu, podczas gdy mysz umożliwia poruszanie się po ekranie i wykonywanie operacji wyboru i manipulacji.
5. **Monitor:** Jest wyjściowym urządzeniem, które wyświetla informacje wizualne dla użytkownika. Może to być ekran LCD, LED lub inna technologia, która przedstawia dane w formie graficznej lub tekstowej.
6. **Karta graficzna:** Odpowiada za generowanie i wyświetlanie grafiki na monitorze. Wspiera przetwarzanie graficzne, jak również może obsługiwać zaawansowane efekty wizualne, 3D, multimedia itp.
7. **Płyta główna:** Jest to centralne ogniwo, na którym zamocowane są wszystkie podzespoły komputera. Zapewnia łączność między różnymi komponentami i umożliwia przesyłanie danych oraz zasilanie między nimi.

Co więcej, komputer może mieć inne urządzenia wejścia/wyjścia, takie jak drukarka, głośniki, kamera internetowa, mikrofon itp., które rozszerzają funkcjonalność i możliwości komunikacji z zewnętrznymi urządzeniami.

Ważne jest, aby pamiętać, że komputery są produktem złożonej inżynierii, której celem jest skuteczne

wykorzystanie elektroniki, technologii programowania i architektury systemów, aby stworzyć wszechstronne narzędzie do przetwarzania danych i wykonywania zadań. A tak naprawdę to chodzi o to, żeby gierki na nich płynnie działały. ☺

A po co w tym wszystkim potrzebny jest system operacyjny?

Obraz 5. Źródło: https://reddit.com

Żeby człowiek mógł działać, potrzebna jest kawa. Żeby komputer mógł działać, potrzebny jest program. Najbardziej podstawowym programem, o którym możemy pomyśleć, jest System operacyjny.

System operacyjny (ang. *Operating System*) to oprogramowanie, które zarządza zasobami komputera i umożliwia użytkownikom interakcję z maszyną. Można go porównać do *"dyrektora operacyjnego"* komputera, który nadzoruje i koordynuje wszystkie działania systemu. Główne zadania tego *"dyrektora"* to:

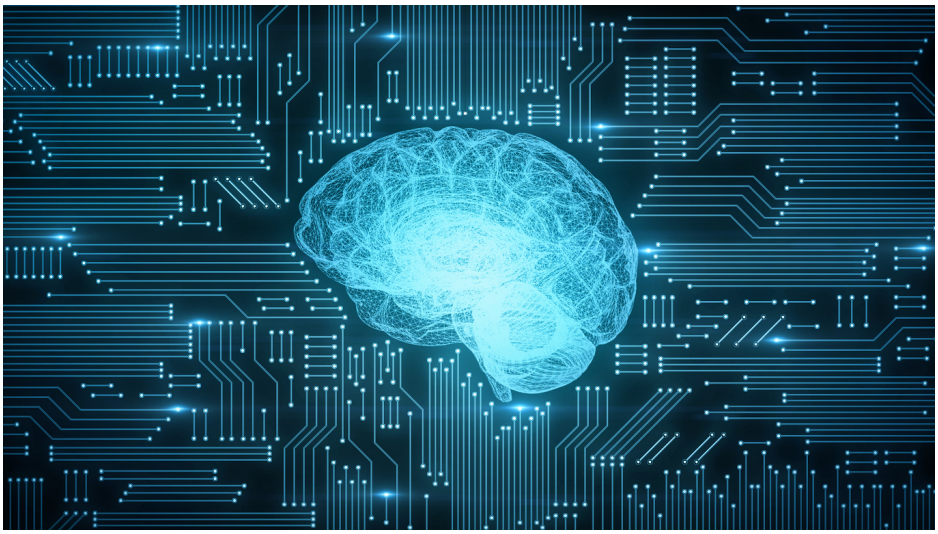
1. **Zarządzanie zasobami:** System operacyjny zarządza zasobami komputera, takimi jak procesor, pamięć, urządzenia wejścia/wyjścia i sieć. Przydziela zasoby użytkownikom i programom, aby mogły one efektywnie działać.
2. **Wykonywanie programów:** System operacyjny umożliwia uruchamianie i wykonywanie programów komputerowych. Zarządza kolejnością i priorytetami procesów, co pozwala na równoczesne wykonywanie wielu programów.
3. **Zarządzanie pamięcią:** System operacyjny kontroluje alokację i zwalnianie pamięci operacyjnej (RAM). Przydziela miejsce dla działających programów i zarządza przenoszeniem danych między pamięcią operacyjną a pamięcią masową.
4. **Zarządzanie plikami:** System operacyjny zarządza strukturą plików i folderów na dyskach twardych i innych nośnikach. Zapewnia operacje odczytu, zapisu, kopiowania i usuwania plików oraz kontroluje dostęp do nich.
5. **Interakcja z użytkownikiem:** System operacyjny zapewnia interfejs użytkownika, który umożliwia

interakcję z komputerem. Może to być interfejs tekstowy (wiersz poleceń) lub graficzny (np. pulpity, ikony, menu), który ułatwia użytkownikowi korzystanie z komputera.

System operacyjny działa w sposób ciągły, monitorując działanie komputera i reagując na zdarzenia. Korzysta z różnych mechanizmów, takich jak harmonogramowanie procesów, zarządzanie pamięcią wirtualną, sterowniki urządzeń i protokoły sieciowe, aby zapewnić sprawną pracę systemu.

W skrócie system operacyjny pełni kluczową funkcję w zarządzaniu zasobami komputera i umożliwia użytkownikom wygodne korzystanie z maszyny. Zapewnia platformę dla innych aplikacji i programów, a także zapewnia bezpieczeństwo, stabilność i efektywność działania całego systemu komputerowego.

Czy komputer myśli?



Obraz 6. Źródło: Computational thinking: a key skill in the 21st century | TrainingZone

Nie, komputer nie myśli w takim sensie, w jaki my, ludzie, rozumiemy proces myślenia. Komputery są maszynami, które operują na podstawie logicznych instrukcji i algorytmów, a ich działanie jest całkowicie oparte na przetwarzaniu informacji według ściśle określonych reguł.

Komputery mogą wykonywać zadania i podejmować operacje na danych z niezwykłą prędkością i precyzją, ale nie mają wiedzy, świadomości ani zdolności do myślenia w sposób abstrakcyjny. Wszystkie operacje, które wykonują, są wynikiem zaprogramowanych instrukcji i manipulacji danych na podstawie tych instrukcji.

Chociaż komputery potrafią wykonywać skomplikowane obliczenia, rozpoznawać wzorce, generować odpowiedzi na zapytania czy analizować dane, to robią to na podstawie precyzyjnych i z góry określonych instrukcji. Brak im subiektywności, emocji i zdolności do samodzielnego tworzenia myśli.

Pojęcie *"myślenia"* jest związane z procesami poznawczymi, które są charakterystyczne dla ludzkiego umysłu. Ludzie mają zdolność do abstrakcyjnego myślenia, kreatywności, rozumienia kontekstu, podejmowania decyzji opartych na wartościach i uczuciach. To jest obszar, który odróżnia nas od maszyn.

Choć komputery są niezwykle przydatne i potężne w przetwarzaniu danych, nie mają zdolności do myślenia w sensie ludzkim. Są narzędziami, które wykonują zadania na podstawie zaprogramowanych instrukcji, ale nie posiadają świadomości ani subiektywnych doświadczeń, które są charakterystyczne dla ludzkiego myślenia. No, chyba, że mówimy o Stefanie.

Zapamiętaj coś, co już padło wcześniej: komputery przechowują i przetwarzają dane, wykonując na nich obliczenia i operacje logiczne, ale robią to na podstawie programów, które są wykonywane dzięki procesorowi i pamięci operacyjnej.

Co to jest to całe programowanie?

Obraz 7. Źródło: https://demotywatory.pl/

Przejdźmy (nareszcie) do wyjaśnienia zagadnienia klucz! Czym że jest to całe programowanie?

Programowanie to proces tworzenia programów (dziękujemy kapitanie oczywisty), czyli zbiorów instrukcji, które określają, co ma się dziać i w jakiej kolejności, żeby osiągnąć jakiś konkretny cel.

Programowanie to proces tworzenia zestawu instrukcji, zwanych programem, które sterują zachowaniem komputera lub innego urządzenia. Programowanie polega na tworzeniu kodu, który składa się z serii instrukcji, wyrażeń i algorytmów, które określają, jakie operacje i obliczenia powinny być wykonane przez komputer.

Programowanie obejmuje pisanie kodu w określonym języku programowania, który jest zrozumiały dla komputera. Języki programowania mogą być różnorodne i różnią się składnią i strukturą, ale wszystkie mają na celu umożliwienie programiście przekazania instrukcji do wykonania przez komputer.

Programowanie może być używane do różnych celów, takich jak tworzenie aplikacji mobilnych, stron internetowych, oprogramowania biurowego, gier komputerowych, systemów operacyjnych i wielu innych. Programiści używają narzędzi programistycznych, takich jak edytory kodu, kompilatory, debuggery, aby pisać, testować i uruchamiać swoje programy.

Programowanie wymaga logicznego myślenia, rozwiązywania problemów i umiejętności analitycznych. Jest to proces twórczy, który pozwala na realizację różnorodnych idei i rozwiązywanie

skomplikowanych zadań za pomocą mocy obliczeniowej komputera.

W dzisiejszym świecie programowanie ma ogromne znaczenie i jest nieodłączną częścią postępu technologicznego. Umożliwia rozwój nowych technologii, automatyzację zadań, analizę danych, komunikację między ludźmi oraz wiele innych dziedzin, wpływając na nasze codzienne życie, sposób pracy oraz zapotrzebowanie na flanelowe koszule dla programistów.

Tylko tyle i aż tyle. Przejdźmy zatem do kolejnej kwestii.

Po co tworzy się programy komputerowe?

A no po to, żebyśmy wszystko, czego potrzebujemy, mieli w zasięgu kliknięcia myszy. Programy komputerowe są pisane w celu rozwiązywania problemów, automatyzowania procesów, ułatwiania pracy i wykonywania różnych zadań za pomocą komputera. Programy te mogą być napisane w różnych językach programowania i wykorzystywać różne technologie, w zależności od celu, dla którego zostały stworzone.

To może jakieś konkrety? Po co tworzy się programy komputerowe?

1. **Rozwiązanie problemów:** Programy komputerowe pomagają w rozwiązywaniu problemów matematycznych, naukowych, inżynieryjnych czy biznesowych, które byłyby zbyt złożone lub czasochłonne do rozwiązania ręcznie.
2. **Automatyzacja:** Programy pozwalają na automatyzację zadań i procesów, zwiększając efektywność i redukując błędy. Przykładami są systemy zarządzania bazami danych, oprogramowanie księgowe czy roboty przemysłowe.
3. **Ułatwianie życia:** Programy komputerowe pomagają ludziom w codziennych zadaniach, takich jak komunikacja, zarządzanie informacją, organizacja pracy czy rozrywka. Przykłady obejmują przeglądarki internetowe, edytory tekstu, aplikacje do zarządzania czasem czy gry komputerowe.
4. **Innowacje technologiczne:** Programowanie pozwala na tworzenie nowych technologii, które zmieniają sposób, w jaki działają różne branże. Przykładami są rozwój sztucznej inteligencji, rozwiązania w zakresie internetu rzeczy (IoT), czy technologie np. blockchain.
5. **Edukacja i badania:** Programy komputerowe są używane w edukacji i badaniach naukowych, pomagając w analizie danych, modelowaniu zjawisk czy symulacjach.
6. **Wsparcie biznesu:** Programy komputerowe są stosowane w wielu aspektach biznesu, takich jak zarządzanie relacjami z klientami (CRM), planowanie zasobów przedsiębiorstwa (ERP) czy analityka biznesowa.
7. **Sztuka i projektowanie:** Programy komputerowe umożliwiają tworzenie cyfrowych dzieł sztuki, animacji, grafiki 3D, efektów specjalnych czy projektowania stron internetowych. Polecamy tutaj zapoznanie się z takimi narzędziami jak np. Midjourney, czy DALL·E.

Tworzenie programów komputerowych może mieć zatem różnorodne cele i wpływa na praktycznie każdy aspekt naszego życia, od codziennych czynności po zaawansowane badania naukowe. W dzisiejszych czasach nawet na nadgarstku możesz nosić program!

W jaki sposób tworzone są programy komputerowe?

Programy komputerowe są tworzone w procesie, który składa się z kilku etapów, często tych etapów jest sporo. Poszczególne metody mogą się różnić w zależności od języka programowania, środowiska pracy czy celów projektu, natomiast ogólnie proces ten nazywa się cyklem życia oprogramowania (ang. *Software Development Life Cycle*) i składa się z następujących kroków:

1. **Planowanie:** Pierwszym krokiem jest określenie celu i wymagań programu. Programiści i projektanci zbierają informacje od klienta lub użytkownika, analizują potrzeby i określają funkcjonalności, jakie program ma posiadać. Tworzone są specyfikacje, diagramy, storyboardy, które pomagają w zrozumieniu i zaplanowaniu struktury programu.
2. **Projektowanie:** Na podstawie zebranych informacji projektanci tworzą projekt programu. Określają architekturę, strukturę danych, interfejs użytkownika i sposób działania programu. Mogą używać różnych narzędzi i technik, takich jak diagramy przepływu danych, diagramy klas, prototypowanie, aby zaplanować szczegóły implementacji.
3. **Implementacja:** W tym etapie programiści przekształcają projekt w kod komputerowy. Używają odpowiedniego języka programowania i narzędzi programistycznych, aby napisać kod, który realizuje wymagania programu. Mogą tworzyć różne moduły, funkcje, klasy, które są odpowiedzialne za konkretne zadania.
4. **Testowanie:** Po napisaniu kodu, program jest poddawany testom, aby sprawdzić, czy działa poprawnie i spełnia założone wymagania. Testy mogą obejmować różne scenariusze, sprawdzanie poprawności działania, wydajności, bezpieczeństwa i interakcji z użytkownikiem. Błędy i niedociągnięcia są identyfikowane i naprawiane.
5. **Wdrażanie:** Po zakończeniu testów, gotowy program jest wdrażany do środowiska docelowego. Może to oznaczać instalację na komputerach użytkowników, udostępnienie w chmurze lub inny sposób dystrybucji. Program może wymagać konfiguracji i integracji z innymi systemami.
6. **Utrzymanie:** Po wdrożeniu programu konieczne jest monitorowanie, wsparcie i utrzymanie. Program może wymagać aktualizacji, poprawek błędów, dodania nowych funkcji lub dostosowania do zmieniających się wymagań. Utrzymywanie programu obejmuje również śledzenie wydajności, bezpieczeństwa i ewentualne reagowanie na problemy zgłaszane przez użytkowników.

Ten proces może być iteracyjny, co oznacza, że programista może wracać do wcześniejszych etapów w celu wprowadzenia zmian, ulepszeń lub naprawienia błędów. W praktyce, tworzenie programów komputerowych często odbywa się w ramach różnych metodologii, takich jak programowanie zwinne (**Agile**), **Scrum**, **Kanban** czy programowanie ekstremalne (**XP**). Metodologie te mają na celu zwiększenie efektywności procesu programowania, lepsze zarządzanie projektem, a także lepszą współpracę w zespole programistów.

Warto również wspomnieć, że do tworzenia programów komputerowych często korzysta się z narzędzi, które ułatwiają pracę, takich jak:

1. **Zintegrowane środowiska programistyczne (IDE):** To narzędzia, które łączą edytor kodu, debugger, kompilator oraz inne funkcje, ułatwiając programowanie. Przykłady to *Visual Studio*, *IntelliJ IDEA* czy *PyCharm*.
2. **Systemy kontroli wersji:** Służą do zarządzania zmianami w kodzie źródłowym, umożliwiając współpracę wielu programistów oraz śledzenie historii zmian. Przykłady to *Git*, *Subversion* czy *Mercurial*.

3. **Narzędzia do automatyzacji budowy i wdrożeń:** Pozwalają na automatyzację procesów kompilacji, testowania i wdrażania oprogramowania. Przykłady to *Jenkins*, *Travis CI* czy *Gradle*.
4. **Biblioteki, frameworki i narzędzia programistyczne:** Ułatwiają tworzenie programów, dostarczając gotowych do użycia komponentów, funkcji czy szablonów. Pozwalają na oszczędność czasu oraz zmniejszenie ryzyka błędów.

Nie przejmuj się nowymi terminami. Po to powstała Zajavka, żeby przeprowadzić Cię za rękę, krok po kroku przez skomplikowany proces nauki programowania. Uwierz nam, że gdy dojdziesz do końca zajavkowej ścieżki, to wiedza zawarta w tej książce będzie dla Ciebie prosta i oczywista.

W zależności od projektu programy komputerowe mogą być tworzone przez pojedynczego programistę, zespół programistów, a nawet przez współpracujące ze sobą firmy. Proces ten jest dynamiczny i ewoluuje wraz z rozwojem technologii, narzędzi oraz potrzeb użytkowników.

Co muszę umieć, żeby zostać programistą?

Obraz 8. Źródło: https://demotywatory.pl/

Aby zostać programistą, po pierwsze musisz lubić siedzieć w piwnicy a po drugie, musisz nie lubić ludzi. A tak naprawdę musisz po prostu nabyć pewne umiejętności i wiedzę, które będą Ci potrzebne do tworzenia oprogramowania. Proste nie?!

Nawet jeżeli wydaje ci się, że programowanie to czysta matematyka — nic bardziej mylnego. Programiści powinni posiadać dużo różnych umiejętności. Przejdźmy teraz przez kilka z nich i zacznijmy od umiejętności *"miękkich"*:

1. **Umiejętności analityczne:** Programiści muszą posiadać umiejętności analityczne, które pozwalają

im rozwiązywać problemy, projektować programy i tworzyć skuteczne rozwiązania.

2. **Kreatywność:** Programowanie wymaga kreatywności i myślenia poza schematami, aby znaleźć rozwiązania, które są wydajne, efektywne i łatwe w obsłudze.
    - Zdolność do pracy zespołowej: Programiści często pracują w zespołach, dlatego ważne jest, aby umieć pracować z innymi programistami, a także z projektantami, testerami i klientami.
3. **Umiejętności rozwiązywania problemów:** Programiści powinni posiadać umiejętności rozwiązywania problemów, które pozwalają im identyfikować problemy, analizować je i znajdować skuteczne rozwiązania.
4. **Systematyczne uczenie się:** Programowanie to ciągły proces nauki, więc ważne jest, aby zacząć od podstawowych pojęć i systematycznie rozwijać swoje umiejętności.
5. **Logiczne myślenie:** Programowanie wymaga logicznego myślenia, czyli umiejętności rozwiązywania problemów i przewidywania skutków różnych działań.
6. **Komunikatywność:** Programiści powinni być w stanie jasno i skutecznie komunikować się z klientami, menedżerami projektów i innymi członkami zespołu.

Oczywiście to, co zostało wymienione to nie wszystko. Programiści muszą również posiadać umiejętności techniczne:

1. **Znajomość podstaw informatyki:** Programiści powinni znać podstawy informatyki, takie jak architektura komputerów, systemy operacyjne, algorytmy, struktury danych i bazy danych.
2. **Algorytmy i struktury danych:** Warto zrozumieć, co to są algorytmy i jak działają różne struktury danych, takie jak tablice, listy i drzewa.
    - Podstawy języka programowania: Istotne są również podstawowe pojęcia związane z językami programowania, takie jak zmienne, typy danych, instrukcje warunkowe i pętle.
3. **Znajomość języków programowania:** Programiści muszą znać co najmniej jeden język programowania, w zależności od specjalizacji i celów programowania. Popularne języki programowania to Java (wiadomo), Python, C++, JavaScript, Ruby, PHP i wiele innych.
4. **Zasady programowania:** Powinieneś/powinnaś znać podstawowe zasady programowania takie jak jasność, prostota, elegancja, poprawność, wydajność i modułowość.
5. **Umiejętności techniczne:** Programiści muszą być zaznajomieni z narzędziami i technologiami używanymi w programowaniu, takimi jak środowiska programistyczne, systemy kontroli wersji, narzędzia do debugowania, frameworki i biblioteki.
6. **Czytanie i pisanie kodu:** Zanim zaczniesz samodzielnie pisać kod, warto nauczyć się czytać i analizować istniejący kod, aby zrozumieć, jak działa i co można z niego wynieść.

I oczywiście, *last but not least*:

Znajomość języka angielskiego! Większość materiałów związanych z programowaniem jest napisana w języku angielskim, dlatego warto znać ten język na poziomie przynajmniej podstawowym, aby łatwiej przyswajać wiedzę.

Obraz 9. Źródło: Akademia Polskiego Filmu

Czyli w sumie wychodzi na to, że żeby zostać programistą, trzeba znać co najmniej jeden język programowania, posiadać podstawową wiedzę z zakresu informatyki, mieć rozwinięte umiejętności analityczne, kreatywność, zdolność do pracy zespołowej. Potrzebne również będą umiejętności techniczne, umiejętności rozwiązywania problemów i komunikatywność. A mówili, że będzie łatwo... i że dużo zapłacą...

Dużo, prawda? Wiedzy jest dużo, przecież programiści nie zarabiają tyle za nic ☺.

Podstawowe pojęcia programistyczne

Jakie pojęcia powinienem znać jako początkujący programista?

Każda branża ma jakieś swoje *"dziwne"* pojęcia i tak samo jest tutaj. Przejdziemy sobie teraz bardzo pobieżnie, przez niektóre z nich. Później wrócimy do tych pojęć i opowiemy o nich bardziej szczegółowo.

Kod źródłowy

Kod źródłowy to **tekst zapisany w jakimś konkretnym języku programowania**. Kod źródłowy to lista instrukcji, które mówią, co ma zrobić komputer. Programista piszący kod źródłowy używa specjalnych narzędzi programistycznych, takich jak np. **edytory kodu**, aby ułatwić sobie pracę, ale równie dobrze mógłby pisać taki kod w notatniku.

Edytor kodu

Edytory kodu to narzędzia programistyczne, które pozwalają programistom pisać kod źródłowy w sposób efektywny (szybciej, sprawniej i wygodniej niż w notatniku). W edytorze kodu programista może **zapisywać** kod, **edytować** go i **przeglądać**. Edytory kodu posiadają funkcjonalności, takie jak

podświetlanie składni (możesz sobie pokolorować różne części programu), wskazywanie błędów i podpowiadanie składni danego języka programowania.

IDE

IDE (**I**ntegrated **D**evelopment **E**nvironment), które są specjalnymi narzędziami programistycznymi, ułatwiającymi programistom tworzenie kodu źródłowego. IDE zawiera zintegrowane **edytory kodu**, **narzędzia do debugowania** (dowiesz się później, co to jest) i **kompilatory**, co pozwala na łatwe tworzenie i testowanie kodu.

Kompilacja

Kolejnym istotnym pojęciem związanym z programowaniem jest **kompilacja**. Kompilacja to proces **przekształcania kodu źródłowego na kod maszynowy**, który może być bezpośrednio wykonywany przez procesor komputera. Proces ten jest wykonywany przez specjalny program zwany **kompilatorem**.

Komputer nie jest w stanie zrozumieć języka programowania, w którym Ty piszesz program (komputer rozumuje trochę inaczej), do tego kod musi zostać skompilowany (czyli zamieniony z języka, który Ty rozumiesz, na język, który rozumie komputer). Komputer jest w stanie wykonać kod skompilowany. Oczywiście opis ten jest bardzo urposzczony, w praktyce to zagadnienie jest bardziej skomplikowane.

Interpretacja

Interpretacja to inny sposób przetwarzania kodu źródłowego, w którym kod jest wykonywany linia po linii przez **interpreter**. Interpreter to program, który **czyta kod źródłowy i natychmiast go wykonuje**, bez konieczności kompilowania go na kod maszynowy.

Różne języki programowania

Ważnym elementem programistycznego świata są różne języki programowania i ich zastosowania. Istnieje wiele języków programowania, takich jak **Python**, **Java**, **C++**, **JavaScript** czy **Ruby**, z których każdy ma swoje unikalne cechy i zastosowania. Dlatego warto zrozumieć, co to jest język programowania i jakie są jego podstawowe elementy.

Język programowania

Język programowania to zestaw reguł i konwencji (składnia języka, tak jak np. francuski), które programista używa do pisania kodu źródłowego. Języki programowania dzielimy na języki niskiego i wysokiego poziomu. Języki niskiego poziomu, takie jak **Asembler**, pozwalają na bezpośrednie korzystanie z procesora komputera, natomiast języki wysokiego poziomu, takie jak **Python**, są bardziej abstrakcyjne i pozwalają na łatwiejsze pisanie kodu (z perspektywy programisty).

Podstawowymi elementami języków programowania są **zmienne, typy danych, instrukcje warunkowe** oraz **pętle**.

Zmienna

Zmienna to pojemnik, który przechowuje wartość, która może się zmieniać w trakcie wykonywania programu. Zmienna może być zdefiniowana przy określeniu typu danych, gdzie typy danych definiują,

jakie wartości mogą być przechowywane w zmiennej, na przykład liczby, tekst czy wartości logiczne (*prawda/fałsz*).

Instrukcje warunkowe

Instrukcje warunkowe pozwalają na **wprowadzenie logiki do programu**, pozwalając na wykonywanie różnych instrukcji, w zależności od spełnienia określonych warunków. Czyli np. *"Jeżeli masz skończone 18 lat, możesz kupić alkohol, jeżeli nie masz to sorry!"*

Pętle

Pętle z kolei pozwalają na **powtarzanie tych samych instrukcji wielokrotnie**, co ułatwia automatyzację powtarzających się zadań. Pętle w programowaniu są jak powtarzanie czynności, które chcesz zrobić wiele razy. Wyobraź sobie, że chcesz podskoczyć 10 razy. Zamiast mówić *"podskocz"* 10 razy z rzędu, możemy powiedzieć: *"podskocz 10 razy"*. Pętla to taka komenda dla komputera, która mówi mu, że ma coś zrobić wielokrotnie - tak, jak podskakiwanie.

Debugowanie

Ważnym aspektem wprowadzenia do programowania jest także nauka **debugowania** kodu. Debugowanie to proces **znajdowania i usuwania błędów** w kodzie źródłowym. W tym celu programista korzysta z narzędzi takich jak **debugger**, który pozwala na stopowanie wykonania programu w wybranym miejscu, a następnie krok po kroku przeglądanie i analizowanie kodu.

Abstrakcja

Abstrakcja w programowaniu to jak myślenie o zabawkach, zamiast o ich poszczególnych częściach. Gdy bawisz się samochodzikiem, nie musisz wiedzieć, jak każde koło, silnik czy drzwi są zbudowane, ani jak działa mechanizm skręcania kół. Ważne jest, że samochodzik jeździ, a Ty się dobrze bawisz.

W programowaniu abstrakcja to sposób, w jaki programiści ukrywają skomplikowane części kodu, tak aby inni mogli skupić się na tym, co program robi, a nie na tym, jak dokładnie to robi. Tak jak w przypadku zabawek, nie musisz znać wszystkich szczegółów, aby się nimi bawić i dobrze się bawić.

Wiemy, że pojawiło się tutaj właśnie wiele nowych pojęć. Pamiętaj, że po przeczytaniu tej książki, nie masz dogłębnie rozumieć każdego z wymienionych pojęć. Masz wiedzieć, w którym kościele dzwoni dzwon, a nie, jak i dlaczego ten dzwon dzwoni ☺.

Powiedzieliśmy sobie o bardzo podstawowych pojęciach w bardzo podstawowym zakresie. Możemy teraz przejść do bardziej szczegółowego omówienia niektórych z wymienionych zagadnień.

Czym jest kod źródłowy programu?

Kod źródłowy programu to **tekstowa reprezentacja programu**, napisana w języku programowania, zgodnie z zasadami składni danego języka. Jest to lista instrukcji, które komputer może odczytać i przetworzyć, aby wykonać określone zadanie.

Kod źródłowy jest pierwotnym źródłem programu i składa się z linii kodu, które są zapisane w pliku tekstowym. Tak, programy można pisać nawet w notatniku, gdyż są to zwykłe pliki tekstowe. To, co jest

w nich napisane, ma już znaczenie. Plik taki zawiera definicje klas, funkcji, zmiennych, wyrażeń i innych elementów języka programowania, które są potrzebne do zrozumienia i wykonywania programu. Wymienione elementy języków programowania poznajesz, ucząc się danego języka programowania, także spokojnie ☺.

Kod źródłowy jest tworzony przez programistę i jest podstawowym elementem w procesie tworzenia oprogramowania. Po napisaniu kodu źródłowego programista musi go skompilować lub zinterpretować, aby powstał plik wykonywalny, który może być uruchomiony na komputerze.

Kod źródłowy jest zwykle przechowywany w systemach kontroli wersji, takich jak *Git* czy *SVN*, co pozwala na śledzenie zmian i udostępnianie kodu w zespołach programistycznych. W ramach ścieżki zajavka poznajemy narzędzie, jakim jest *Git*, dlatego w tym momencie powinno Ci wystarczyć, że *Git* służy tak jakby do *"trzymania Twojego kodu w chmurze"*.

Czym jest kod maszynowy?

Było o kodzie źródłowym, to teraz pora na kod maszynowy ☺.

Kod maszynowy to najniższy poziom abstrakcji w języku komputera, czyli sekwencja instrukcji zapisana w formacie binarnym (zera i jedynki), które są bezpośrednio wykonywane przez procesor. Każda instrukcja składa się z ciągu zer i jedynek, które są interpretowane przez procesor jako konkretne działania, takie jak pobieranie wartości z pamięci, wykonywanie operacji arytmetycznych, czy przesyłanie danych do urządzeń wejścia/wyjścia (myszka, klawiatura, ekran).

Kod maszynowy jest generowany przez kompilatory lub interpretatory, które tłumaczą kod źródłowy napisany w językach programowania na instrukcje zrozumiałe przez procesor. Proces generowania kodu maszynowego z kodu źródłowego nazywa się kompilacją lub interpretacją, w zależności od tego, jakie narzędzie jest używane.

Kod maszynowy jest bardzo trudny do czytania i pisania przez człowieka, ponieważ składa się on z ciągu zer i jedynek. Dlatego większość programistów korzysta z języków programowania wysokiego poziomu, które pozwalają na pisanie kodu w bardziej czytelny i zrozumiały sposób. Jednakże, kod maszynowy jest nadal istotny, ponieważ jest ostatecznym wynikiem pracy kompilatora lub interpretora, a także jest używany w procesie debugowania i optymalizacji kodu.

Można zatem powiedzieć, kod maszynowy to sekwencja instrukcji zapisana w binarnym formacie, która jest bezpośrednio wykonywana przez procesor. Jest on generowany przez kompilatory lub interpretatory i jest ostatecznym wynikiem pracy programisty.

Czym jest Assembler?

Jest też coś takiego jak **Assembler**, który jest językiem programowania **niskiego poziomu**, który służy do tworzenia programów komputerowych, a konkretnie do pisania kodu maszynowego. Kod maszynowy to zestaw instrukcji, które komputer może bezpośrednio wykonać (tak, jak powiedzieliśmy w poprzednim paragrafie).

Assembler umożliwia programistom pisanie programów w sposób bardziej zrozumiały dla człowieka niż bezpośrednie pisanie kodu maszynowego. W Assemblerze każda instrukcja języka odpowiada jednej instrukcji maszynowej, co umożliwia programistom bezpośrednie kontrolowanie sprzętu

komputerowego.

Assembler jest często stosowany do tworzenia programów, które wymagają maksymalnej wydajności, ponieważ programista może wtedy zoptymalizować kod do bezpośredniego wykorzystania przez procesor. Jednak pisanie kodu w Assemblerze jest znacznie trudniejsze niż w innych językach programowania, ponieważ wymaga dogłębnej znajomości architektury procesora i kodowania instrukcji maszynowych.

Dlatego my na Zajavce uczymy Javy, czyli języka programowania, który jest językiem wyższego poziomu i jest bardziej zrozumiały dla ludzi. No, może poza Stefanem.

Czym jest kompilacja i kompilator?

Nie, nie kombinacja. Ani nie komplikacja. (Chociaż może trochę...) Było pobieżnie, teraz wyjaśnijmy to sobie dokładniej ☺.

Kompilacja to proces **tłumaczenia kodu źródłowego** napisanego w języku programowania na **kod maszynowy**, czyli instrukcje zrozumiałe dla procesora komputera. Proces ten jest wykonywany przez specjalne programy zwane kompilatorami. **Kompilator analizuje kod źródłowy** i tworzy plik wykonywalny, który może być **uruchomiony bezpośrednio na komputerze**.

Kompilacja jest procesem bardzo ważnym w procesie tworzenia oprogramowania, ponieważ pozwala na stworzenie pliku wykonywalnego, który może być uruchomiony na różnych komputerach. Dzięki temu, że plik wykonywalny jest już przetłumaczony na kod maszynowy, procesor komputera może go bezpośrednio wykonywać, co przyspiesza działanie programu.

Kompilacja składa się z kilku etapów. Pierwszym etapem jest analiza kodu źródłowego, w której kompilator sprawdza poprawność składniową kodu i buduje strukturę programu. Następnie kompilator dokonuje optymalizacji kodu, usuwając zbędne fragmenty kodu i zastępując niektóre instrukcje bardziej efektywnymi. W kolejnym etapie kompilator generuje kod maszynowy, który jest zapisywany w pliku wykonywalnym.

Kompilacja jest procesem niezbędnym do uruchomienia programów napisanych w językach programowania, takich jak np. C++ czy Java. Bez kompilacji nie byłoby możliwe przetłumaczenie kodu źródłowego na kod maszynowy, który jest niezbędny do uruchomienia programu na komputerze.

Można zatem powiedzieć, że **kompilacja to proces tłumaczenia kodu źródłowego na kod maszynowy**, a **kompilator to program, który wykonuje ten proces**. Kompilacja jest niezbędna do uruchomienia programów napisanych w językach programowania i pozwala na stworzenie pliku wykonywalnego, który może być uruchomiony na różnych komputerach. Czyli jednak trochę kombinacja i komplikacja.

Różne języki programowania

Obraz 10. Źródło: https://paczaizm.pl/

W poprzedniej części tej książki, co jakiś czas przewija się stwierdzenie: *"język programowania"*. Z innymi ludźmi możesz porozumiewać się w różnych językach, np. angielski, francuski, portugalski. W przypadku komputera polecenia wydawane są przy wykorzystaniu różnych języków programowania.

Czym różnią się od siebie języki programowania?

Nazwą. Przejdźmy dalej. No dobra, czymś tam się jeszcze różnią...

Na świecie zostało opracowanych bardzo dużo języków programowania, a Ty możesz nauczyć się wielu z nich. Jednym z ulubionych języków Karola (jako ciekawostka przyrodnicza) jest Brainfuck, o którym możesz poczytać np. na Wikipedii.

Języki programowania różnią się od siebie w wielu aspektach i to właśnie na tym chcielibyśmy się skupić w tym paragrafie. Więc czym się one różnią?

1. **Składnia:** każdy język programowania ma swoją własną składnię, czyli zestaw reguł, które określają, jak pisany jest kod w danym języku. Składnia może mieć wpływ na czytelność kodu i łatwość pisania.
2. **Paradygmat programowania:** języki programowania mogą mieć różne paradygmaty programowania, takie jak proceduralne, obiektowe, funkcyjne, logiczne itp. Paradygmat określa, jakie są podstawowe sposoby organizowania kodu i rozwiązywania problemów.
3. **Biblioteki i narzędzia:** każdy język programowania ma swoje własne biblioteki i narzędzia, które ułatwiają programowanie w danym języku. Dostępne biblioteki i narzędzia mogą mieć wpływ na to, jak łatwo i efektywnie można pisać kod.

4. **Wydajność:** różne języki programowania mogą mieć różną wydajność, co oznacza, że niektóre języki są szybsze lub bardziej wydajne niż inne w wykonaniu określonych zadań.
5. **Zastosowanie:** różne języki programowania są stosowane do różnych zastosowań. Na przykład, język JavaScript jest często stosowany w tworzeniu interaktywnych aplikacji internetowych, podczas gdy język C++ jest często stosowany w tworzeniu gier komputerowych.

Dlatego właśnie języki programowania, w których realizowane są projekty, są wybierane w zależności od potrzeb. Wybór języka programowania do realizacji projektu zależy od rodzaju zadania, jakie chcemy wykonać, naszego doświadczenia jako programisty, preferencji i wielu różnych czynników, jak np. podejście danej firmy do różnych technologii.

Swoje projekty możesz oczywiście realizować w dowolnym języku programowania. Jeżeli decydujesz się już, że chcesz nauczyć się programować, warto zapoznać się z ilością ofert pracy na polskim rynku w danym języku programowania. Ale wiadomo, że Java najlepsza.

Jakie mogą być przykładowe zastosowania języków programowania?

Czyli do czego nam to potrzebne? Wiemy już, czym języki programowania mogą się od siebie różnić. A jakie są różnice, jeżeli chodzi o zastosowania różnych języków programowania?

1. **Java**: używana w tworzeniu aplikacji dla różnych platform, w tym aplikacji mobilnych na Androida, aplikacji internetowych, systemów zarządzania treścią, gier itp.
2. **Python**: popularny w analizie danych, sztucznej inteligencji i uczeniu maszynowym, a także w tworzeniu skryptów, automatyzacji zadań, aplikacji internetowych, gier itp.
3. **JavaScript**: stosowany do tworzenia dynamicznych stron internetowych i aplikacji internetowych, w tym interaktywnych interfejsów użytkownika, animacji, gier, wtyczek przeglądarkowych itp.
4. **C++**: używany w tworzeniu aplikacji desktopowych, gier komputerowych, systemów operacyjnych, a także w niektórych zastosowaniach w przemyśle, takich jak sterowanie robotami czy sterowanie procesami produkcyjnymi.
5. **Swift**: język programowania opracowany przez Apple, stosowany w tworzeniu aplikacji na iOS i MacOS, w tym w aplikacjach mobilnych, grach, narzędziach programistycznych, aplikacjach multimedialnych itp.
6. **Ruby**: stosowany w tworzeniu stron internetowych, aplikacji internetowych, narzędzi programistycznych i systemów zarządzania treścią.
    - PHP: używany w tworzeniu stron internetowych i aplikacji internetowych, w tym systemów zarządzania treścią, sklepów internetowych, systemów e-learningowych itp.
7. **Kotlin**: stosowany w tworzeniu aplikacji na Androida, w tym w grach, aplikacjach mobilnych, narzędziach programistycznych itp.
8. **C#**: używany w tworzeniu aplikacji desktopowych, gier komputerowych, narzędzi programistycznych, aplikacji internetowych i innych zastosowaniach.
9. **Go**: stosowany w tworzeniu aplikacji internetowych, narzędzi programistycznych, mikrousług i innych zastosowaniach, a także w przetwarzaniu danych i analizie.

Mamy nadzieję, że udało Ci się zauważyć, że wiele języków programowania może być wykorzystanych

do realizacji tego samego celu. Dlatego podobne projekty mogą być pisane w różnych językach programowania. Inaczej mówiąc, sklep internetowy można napisać w Javie, Pythonie lub PHP, natomiast decyzję, który język programowania wybrać podejmują już tutaj twórcy.

Czym różni się Java od JavaScript?

Obraz 11. Źródło: https://devhumor.com/

Jest to dosyć popularna kwestia, która spędza sen z powiek laikom. Przecież brzmią prawie tak samo, to pewnie ten sam język, nie? No nie!

Odpowiedź na to pytanie (czym różni się Java od JavaScript?) w języku angielskim jest bardziej opisowa: *"Comparing Java with JavaScript is like comparing car with carpet"*. Po polsku moglibyśmy powiedzieć: *"Porównywać Java i JavaScript to jak porównywać ser i serwetkę."* Spróbujemy natomiast wyjaśnić różnice między tymi językami bardziej opisowo.

1. **Typ języka:** Java jest językiem programowania typu kompilowanego, co oznacza, że kod źródłowy jest kompilowany na kod maszynowy przed uruchomieniem, podczas gdy JavaScript jest językiem typu skryptowego, co oznacza, że kod jest interpretowany przez przeglądarkę internetową w czasie rzeczywistym.
2. **Zastosowanie:** Java jest często stosowana w tworzeniu aplikacji desktopowych, aplikacji mobilnych na Androida, systemów zarządzania treścią i innych systemów o dużym stopniu złożoności, natomiast JavaScript jest często stosowany w tworzeniu dynamicznych stron internetowych, interaktywnych interfejsów użytkownika, gier, wtyczek przeglądarkowych i aplikacji internetowych.
3. **Składnia:** Java i JavaScript mają różną składnię i różne struktury języka. Java jest językiem obiektowym, w którym obiekty są kluczowym elementem programowania, natomiast JavaScript jest językiem skryptowym, w którym funkcje są kluczowymi elementami programowania.

4. **Środowisko uruchomieniowe:** Java wymaga zainstalowania w systemie środowiska uruchomieniowego Java (JRE) lub w pełni funkcjonalnej wersji rozwojowej (JDK), aby móc uruchamiać aplikacje, podczas gdy JavaScript uruchamia się w przeglądarce internetowej bez potrzeby dodatkowych narzędzi.

Java i JavaScript to dwa różne języki programowania o różnych zastosowaniach, strukturach i składni. Mimo że mają podobne nazwy, to są to zupełnie odmienne języki programowania.

Zastosowania języków programowania

Możesz się teraz zastanawiać: *"Po cholerę oni znowu mówią o zastosowaniach, skoro już wcześniej ten temat został poruszony?"*. Teraz chcemy przedstawić to samo zagadnienie z innej strony.

Czym jest aplikacja webowa?

Prawda jest taka, że korzystasz z aplikacji webowych często i dużo. Jeżeli uruchomisz przeglądarkę i wejdziesz na Onet — aplikacja webowa, dalej przejdziesz na Instagram — to też aplikacja webowa, a YouTube — to też aplikacja webowa. Co więcej, Google Docs to również aplikacja webowa. Co to w takim razie jest?

Aplikacja webowa to oprogramowanie, które jest dostępne przez przeglądarkę internetową i nie wymaga instalacji na komputerze lub urządzeniu użytkownika. Aplikacje webowe działają na serwerze, a dane są przesyłane przez Internet do przeglądarki użytkownika za pomocą protokołów sieciowych, takich jak HTTP (HyperText Transfer Protocol).

Aplikacje webowe składają się zazwyczaj z dwóch głównych komponentów:

- **Frontend (klient)**: to część aplikacji, która jest wyświetlana w przeglądarce użytkownika. Frontend odpowiada za interfejs użytkownika (UI) oraz interakcję z użytkownikiem. Technologie używane do tworzenia frontendu to HTML (HyperText Markup Language), CSS (Cascading Style Sheets) oraz JavaScript.
- **Backend (serwer)**: to część aplikacji, która działa na serwerze i odpowiada za przetwarzanie żądań od użytkowników, zarządzanie danymi oraz komunikację z bazami danych. Backend może być napisany w różnych językach programowania, takich jak Python, Java, PHP, Ruby czy Node.js, i korzystać z różnych frameworków, takich jak Django, Flask, Spring, Laravel czy Ruby on Rails.

Za moment przejdziemy do głębszego wyjaśnienia, czym jest frontend i backend. (SPOILER — Nic związanego z tenisem!)

Aplikacje webowe są bardzo popularne ze względu na ich łatwość dostępu, łatwość aktualizacji oraz wsparcie dla różnych urządzeń i przeglądarek.

Czym jest aplikacja desktopowa?

Może spotkałeś/-aś się wcześniej ze stwierdzeniem: *"aplikacja desktopowa"*?

Aplikacja desktopowa, inaczej nazywana aplikacją komputerową, to oprogramowanie zaprojektowane do pracy na komputerach stacjonarnych lub laptopach. W przeciwieństwie do aplikacji webowych,

aplikacje desktopowe muszą być zainstalowane na urządzeniu użytkownika, aby mogły działać (czyli apek webowych nie instalujesz, a desktopowe już tak). Aplikacje te zwykle oferują lepsze wykorzystanie zasobów systemowych, wyższą wydajność i dostęp do funkcji systemowych, które nie są dostępne w aplikacjach webowych.

Aplikacje desktopowe mogą być tworzone za pomocą różnych języków programowania i narzędzi, w zależności od platformy docelowej (Windows, macOS, Linux) oraz potrzeb aplikacji. Przykłady języków i frameworków używanych do tworzenia aplikacji desktopowych to:

1. C# i .NET Framework (Windows)
2. C++ i Qt
3. Java i JavaFX
4. Python i PyQt
5. Swift i Cocoa
6. Electron

Przykłady popularnych aplikacji desktopowych to: Microsoft Office, Adobe Photoshop, Visual Studio Code, Slack, czy Spotify.

Warto tutaj wspomnieć, że wraz z postępem technologii, niektóre aplikacje desktopowe są przenoszone do przestrzeni webowej, oferując równocześnie wersje desktopowe i webowe, aby sprostać różnym wymaganiom użytkowników.

Czym różni się aplikacja webowa od desktopowej?

Wiemy już, czym jest aplikacja webowa i czym jest aplikacja desktopowa. Czym w takim razie się one od siebie różnią?

Aplikacje webowe i desktopowe różnią się pod względem architektury, sposobu dystrybucji, dostępu, wydajności oraz możliwości interakcji z systemem operacyjnym. Oto główne różnice między tymi dwoma rodzajami aplikacji:

1. **Dostęp i instalacja:**
    - Aplikacje webowe są dostępne przez przeglądarkę internetową, bez konieczności instalacji na urządzeniu użytkownika. Wystarczy mieć dostęp do Internetu, aby móc z nich korzystać.
    - Aplikacje desktopowe wymagają instalacji na komputerze lub laptopie. Użytkownik musi pobrać i zainstalować oprogramowanie na swoim urządzeniu, aby móc z niego korzystać.
2. **Platforma i kompatybilność:**
    - Aplikacje webowe są zazwyczaj wieloplatformowe, co oznacza, że mogą działać na różnych systemach operacyjnych, takich jak Windows, macOS, czy Linux, pod warunkiem, że użytkownik ma zainstalowaną odpowiednią przeglądarkę internetową.
    - Aplikacje desktopowe są często tworzone specjalnie dla określonej platformy (np. Windows, macOS, czy Linux) i mogą nie działać na innych systemach operacyjnych bez modyfikacji.
3. **Wydajność:**
    - Aplikacje webowe mogą być mniej wydajne niż aplikacje desktopowe, ponieważ są ograniczone

przez przeglądarkę internetową i prędkość połączenia internetowego. Ponadto, przetwarzanie danych może odbywać się na serwerze, co może wpłynąć na opóźnienia i czas reakcji.

- Aplikacje desktopowe mają zazwyczaj lepszą wydajność, ponieważ są zainstalowane na urządzeniu użytkownika i mają bezpośredni dostęp do zasobów systemowych, co pozwala na szybsze przetwarzanie danych.

4. **Dostęp do funkcji systemu operacyjnego:**
   - Aplikacje webowe są ograniczone przez przeglądarkę internetową, co oznacza, że mają mniej możliwości dostępu do funkcji systemowych, takich jak pliki, urządzenia peryferyjne czy powiadomienia systemowe.
   - Aplikacje desktopowe mają większą swobodę w dostępie do funkcji systemu operacyjnego, co pozwala na tworzenie bardziej zaawansowanych i zintegrowanych z systemem aplikacji.
5. **Aktualizacje i utrzymanie:**
   - Aplikacje webowe są łatwiejsze do aktualizacji i utrzymania, ponieważ wszelkie zmiany są wprowadzane na serwerze, a użytkownik zawsze korzysta z najnowszej wersji aplikacji. Nie ma potrzeby instalowania aktualizacji na urządzeniach użytkowników.
   - Aplikacje desktopowe wymagają, aby użytkownik zainstalował nową wersję oprogramowania, gdy pojawi się aktualizacja. W związku z tym proces aktualizacji może być bardziej czasochłonny i skomplikowany.
6. **Bezpieczeństwo:**
   - Aplikacje webowe mogą być narażone na różnego rodzaju ataki, takie jak ataki XSS, CSRF czy SQL Injection, ponieważ dane przesyłane są przez Internet. Jednak odpowiednie praktyki programistyczne i zabezpieczenia serwera mogą pomóc w minimalizacji tych zagrożeń.
   - Aplikacje desktopowe są uważane za bezpieczniejsze, ponieważ dane są przetwarzane lokalnie, a ataki są trudniejsze do przeprowadzenia. Niemniej jednak, aplikacje desktopowe również mogą być narażone na ataki, jeśli nie są odpowiednio zabezpieczone, np. przez malware czy wirusy.
7. **Połączenie internetowe:**
   - Aplikacje webowe zazwyczaj wymagają stałego połączenia z Internetem, aby korzystać z pełnej funkcjonalności. W przypadku braku połączenia, niektóre funkcje mogą być niedostępne lub ograniczone.
   - Aplikacje desktopowe zwykle nie wymagają połączenia z Internetem do działania, chyba że korzystają z usług internetowych lub potrzebują aktualizacji. W takim przypadku, aplikacja może działać bezproblemowo nawet bez dostępu do sieci.

Można zatem powiedzieć, że aplikacje webowe i desktopowe różnią się pod względem architektury, dystrybucji, wydajności oraz możliwości dostępu do funkcji systemowych. Wybór między nimi zależy od potrzeb konkretnego projektu, oczekiwań użytkowników oraz dostępnych zasobów.

W skrócie — potrzebujesz coś mieć pod ręką niezależnie od połączenia z internetem na jednym urządzeniu? Aplikacja desktopowa. Potrzebujesz mieć dostęp do czegoś z różnych urządzeń za pomocą połączenia z internetem? Aplikacja webowa.

Jakie aplikacje są teraz bardziej popularne, webowe czy desktopowe?

W sumie może to nie jest takie oczywiste, dlatego warto byłoby wyjaśnić, jakiego rodzaju aplikacje będziesz często tworzyć w przyszłości. Stąd zadaliśmy sobie pytanie: *"który rodzaj aplikacji jest bardziej popularny?"*

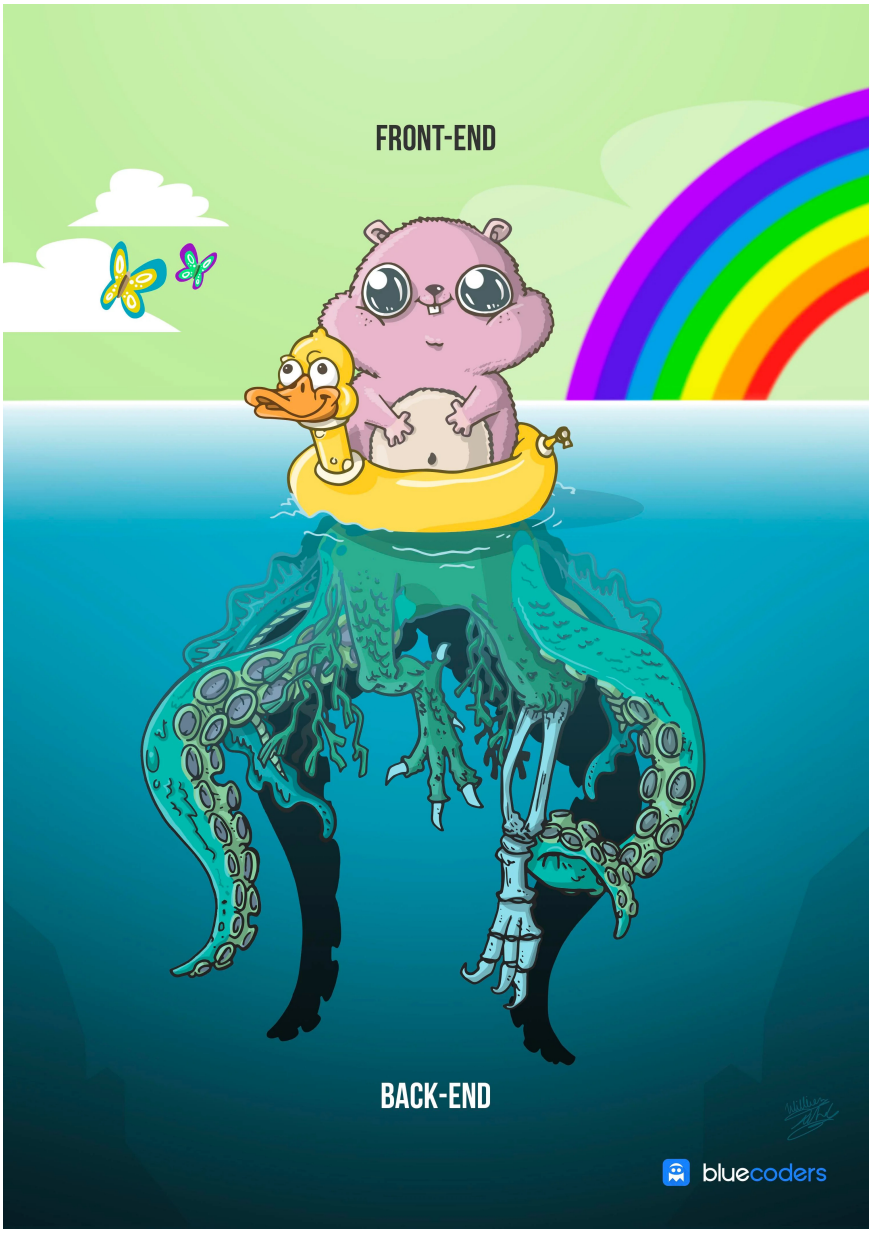
Trudno jest jednoznacznie określić, które aplikacje są obecnie bardziej popularne, ponieważ popularność różnych rodzajów aplikacji zależy od kontekstu, potrzeb użytkowników oraz specyfiki danej dziedziny. Niemniej jednak, w ostatnich latach obserwuje się tendencję wzrostu popularności aplikacji webowych, głównie ze względu na następujące czynniki:

1. **Wieloplatformowość:** Aplikacje webowe działają na różnych systemach operacyjnych i urządzeniach, co sprawia, że są bardziej uniwersalne i łatwiejsze w utrzymaniu.
2. **Łatwość dostępu:** Aplikacje webowe nie wymagają instalacji na urządzeniu użytkownika, co ułatwia ich dystrybucję i aktualizacje. Wystarczy mieć dostęp do Internetu i przeglądarkę internetową, aby móc z nich korzystać.
3. **Rozwój technologii internetowych:** Postęp w technologiach webowych, takich jak HTML5, CSS3, JavaScript oraz WebAssembly, pozwala na tworzenie bardziej zaawansowanych i wydajnych aplikacji webowych, które mogą konkurować z aplikacjami desktopowymi pod względem funkcjonalności i wydajności.
4. **Chmura:** Rosnąca popularność usług chmurowych sprawia, że coraz więcej aplikacji i usług jest dostępnych przez przeglądarkę internetową, co zwiększa zainteresowanie aplikacjami webowymi.
5. **Przenośność danych:** Aplikacje webowe często oferują możliwość przechowywania danych w chmurze, co pozwala na łatwy dostęp do nich z różnych urządzeń oraz współpracę między użytkownikami.

Mimo tych trendów, aplikacje desktopowe nadal są popularne w niektórych dziedzinach, gdzie wymagana jest większa wydajność, dostęp do funkcji systemowych lub lepsza kontrola nad oprogramowaniem. Przykłady takich dziedzin to: edycja grafiki, tworzenie muzyki, projektowanie 3D, czy programowanie.

Ostatecznie, popularność aplikacji webowych i desktopowych zależy od potrzeb użytkowników oraz specyfiki danej dziedziny. W niektórych przypadkach, hybrydowe podejście, które łączy zalety obu typów aplikacji (np. Electron, Progressive Web Apps), może być najlepszym rozwiązaniem.

Czym jest backend?



Obraz 12. Źródło: https://reddit.com

Wcześniej poruszyliśmy już lekko to zagadnienie, poświęćmy jednak jeszcze chwilkę, żeby wyjaśnić je bardziej dogłębnie.

Backend to termin używany w kontekście rozwoju oprogramowania, który odnosi się do warstwy serwerowej aplikacji lub systemu informatycznego. Jest to część systemu odpowiedzialna za obsługę logiki biznesowej, przetwarzanie danych, komunikację z bazami danych i integrację z innymi systemami czy usługami.

W kontekście aplikacji internetowych backend współpracuje z frontendem, który jest odpowiedzialny za interfejs użytkownika i prezentację danych. Gdy użytkownik wprowadza dane lub wykonuje jakąś akcję w interfejsie aplikacji (frontend), te dane są przesyłane do backendu, który przetwarza żądanie, wykonuje odpowiednie operacje na danych i zwraca wyniki do frontendu, aby je wyświetlić użytkownikowi.

Backend może być napisany w różnych językach programowania, takich jak Python, Java, Ruby, PHP, C#

czy JavaScript (Node.js). Wybór języka i technologii zależy od wymagań projektu, preferencji zespołu deweloperskiego i innych czynników.

Czym jest frontend?

Obraz 13. Źródło: https://twitter.com/thetechtopz

Oprócz backendu, aplikacja webowa jest zbudowana również z frontendu.

Frontend to część aplikacji lub strony internetowej, z którą użytkownik bezpośrednio się komunikuje. Obejmuje to interfejs użytkownika (UI), design i zachowania związane z interakcją, które są widoczne dla użytkownika końcowego. Frontend to warstwa, która jest odpowiedzialna za prezentację danych i wygląd aplikacji.

W kontekście rozwoju oprogramowania frontend odnosi się do technologii i narzędzi używanych do tworzenia interfejsów użytkownika. Główne technologie stosowane w tworzeniu frontendu to HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) i JavaScript. HTML definiuje strukturę i zawartość strony, CSS określa jej wygląd i styl, a JavaScript pozwala na interaktywne zachowanie strony.

Frontend jest często przeciwstawiany backendowi, który odpowiada za logikę biznesową i zarządzanie danymi w aplikacji. Backend zazwyczaj komunikuje się z frontendem za pomocą interfejsów programistycznych aplikacji (API) w celu przekazywania danych i informacji między warstwami aplikacji.

Czym różni się frontend od backendu?

Możemy teraz przejść do porównania frontendu i backendu. Zresztą, jak widzisz, my w tej książce często porównujemy ze sobą różne kwestie.

Frontend i backend to dwa główne komponenty aplikacji webowej, które odpowiadają za różne aspekty działania aplikacji. Oto główne różnice między nimi:

1. **Interakcja z użytkownikiem:**
   - Frontend to część aplikacji, która jest bezpośrednio widoczna dla użytkownika i z którą użytkownik może wchodzić w interakcje. Obejmuje to interfejs użytkownika (UI), układ graficzny, animacje i zachowania elementów na stronie.
   - Backend to część aplikacji, która działa w tle, na serwerze, i nie jest widoczna dla użytkownika. Odpowiada za przetwarzanie żądań od frontendu, zarządzanie danymi oraz komunikację z bazami danych i innymi usługami.
2. **Technologie:**
   - Frontend korzysta z technologii webowych, takich jak HTML (HyperText Markup Language) do tworzenia struktury strony, CSS (Cascading Style Sheets) do formatowania i stylizacji oraz JavaScript do dodawania interaktywności i dynamicznych elementów.
   - Backend może być napisany w różnych językach programowania, takich jak Python, Java, PHP, Ruby czy Node.js (JavaScript), i korzystać z różnych frameworków, takich jak Django, Flask, Spring, Laravel czy Ruby on Rails. Backend również komunikuje się z bazami danych, takimi jak MySQL, PostgreSQL, MongoDB czy SQL Server, w celu przechowywania i zarządzania danymi.
3. **Architektura:**
   - Frontend działa na stronie klienta, czyli w przeglądarce internetowej użytkownika, i przetwarza dane lokalnie na urządzeniu użytkownika.
   - Backend działa na serwerze, gdzie przetwarza żądania od frontendu, wykonuje operacje na danych oraz komunikuje się z bazami danych i innymi usługami.
4. **Rola:**
   - Frontend odpowiada za prezentację danych i zapewnienie przyjaznego i atrakcyjnego interfejsu użytkownika, który ułatwia korzystanie z aplikacji.
   - Backend odpowiada za logikę biznesową aplikacji, przetwarzanie danych, autoryzację i autentykację użytkowników, zarządzanie zasobami oraz obsługę błędów i wyjątków.

Można zatem powiedzieć, że frontend i backend są dwoma komponentami aplikacji webowej, które działają razem, aby dostarczyć pełne funkcjonalności aplikacji. Frontend jest odpowiedzialny za interakcje z użytkownikiem i prezentację danych, podczas gdy backend zajmuje się przetwarzaniem danych, logiką biznesową i komunikacją z bazami danych.

W jaki sposób tworzone są gry komputerowe?

Obraz 14. Źródło: https://twitter.com/TheGDWC

My jako twórcy tej książki swoim serduszkiem jesteśmy bardziej związani z tworzeniem aplikacji webowych, ale nie mogło tutaj zabraknąć przynajmniej wzmianki o tym, w jaki sposób tworzone są gry komputerowe. Oczywiście z Twojej perspektywy jest to rozrywka, ale przygotowanie takiej rozrywki wymaga często miesięcy planowania i pracy.

Tworzenie gier jest złożonym procesem, który często wymaga pracy zespołów składających się z różnych specjalistów, takich jak projektanci, programiści, artyści czy dźwiękowcy. Proces ten można podzielić na kilka etapów:

1. **Koncepcja i planowanie:** W tym etapie twórcy gier opracowują pomysł na grę, ustalają jej gatunek, styl, fabułę, postacie, mechaniki rozgrywki oraz cel demograficzny. Powstaje dokument zawierający plan gry, który jest wykorzystywany jako przewodnik dla zespołu.
2. **Projektowanie:** Projektanci gry tworzą plany i szkice poziomów, postaci, przedmiotów, interfejsów użytkownika, animacji i innych elementów. W tym czasie również opracowywane są systemy sterowania, mechaniki rozgrywki i zasady.
3. **Programowanie:** Programiści gry piszą kod, który pozwala na funkcjonowanie gry zgodnie z założonymi mechanikami i regułami. Wykorzystują oni różne języki programowania oraz silniki gier (np. Unity, Unreal Engine, Godot), które umożliwiają szybsze tworzenie gier dzięki gotowym narzędziom i technologiom.
4. **Tworzenie grafiki i dźwięków:** Artyści i projektanci dźwięku pracują nad grafiką, animacjami, efektami wizualnymi, muzyką i efektami dźwiękowymi. Te elementy są niezbędne, aby stworzyć atrakcyjne wizualnie i dźwiękowo doświadczenie dla gracza.
5. **Integracja i testowanie:** Wszystkie elementy gry są łączone, a programiści integrują kod, grafikę i dźwięki. Następnie gra jest testowana pod kątem błędów, problemów związanych z wydajnością, balansem rozgrywki oraz ewentualnych poprawek. Testy mogą być prowadzone przez zespół deweloperski, testerów gier lub graczy w ramach beta-testów.
6. **Optymalizacja i debugowanie:** Na tym etapie wszelkie błędy są identyfikowane i naprawiane, a gra jest optymalizowana pod względem wydajności, aby działała płynnie na różnych platformach i

konfiguracjach sprzętowych.

7. **Marketing i dystrybucja:** Przed i po wydaniu gry, twórcy promują ją w mediach społecznościowych, na stronach internetowych, na konferencjach branżowych i innych kanałach. Gra jest dystrybuowana za pomocą platform cyfrowych (np. Steam, PlayStation Store, Xbox Store) lub na nośnikach fizycznych.
8. **Wsparcie i aktualizacje:** Po wydaniu gry, twórcy mogą udzielać wsparcia technicznego, naprawiać błędy, wprowadzać ulepszenia oraz dodawać nową zawartość poprzez aktualizacje lub rozszerzenia (DLC). Utrzymywanie aktywnego wsparcia dla gry po jej wydaniu jest ważne, aby zadowolić graczy, naprawić ewentualne problemy i utrzymać popularność gry na dłuższy czas.

Tworzenie gier to złożony i czasochłonny proces, który często wymaga współpracy między różnymi specjalistami. Zrozumienie poszczególnych etapów procesu tworzenia gier może pomóc w lepszym zrozumieniu, jak są one projektowane, rozwijane i utrzymywane. A wszystko po to, żeby dwunastolatkowie mogli bez przeszkód obrażać nasze mamy podczas grania przez sieć. ☺

Koncepcje w językach programowania

Można powiedzieć, że na tym etapie rozumiesz już mniej więcej: *"o co w tym programowaniu chodzi"*. Przyszła najwyższa pora na wyjaśnienie bardzo podstawowych zagadnień, z którymi zetkniesz się, jak tylko zaczniesz uczyć się swojego pierwszego języka programowania.

Każde pytanie w tej sekcji omówimy w taki sposób, że zaczniemy od wyjaśnienia danego zagadnienia tak, jakbyśmy tłumaczyli je osobie dorosłej. Na koniec każdego pytania będziemy próbowali wytłumaczyć dane zagadnienie tak, jakbyśmy tłumaczyli je dziecku, czyli krócej i prościej, z pominięciem wielu aspektów, no i trochę innym językiem.

Chcemy w tym miejscu przypomnieć, że w ramach tej książki będą pojawiały się fragmenty kodu, które nie będą na tym etapie zrozumiane przez Ciebie w 100% i to jest normalne. Szczegółowe wyjaśnienia poruszanych w tej książce zagadnień będą w kursie Zajavka. Ta książka ma Ci tylko zajawić temat i dać ogólne poczucie, że wiesz, w którym kościele dzwonią dzwony.

Dlatego nie oczekuj od siebie, że po przeczytaniu tej książki będziesz wszystko pamiętać i rozumieć. To przyjdzie z czasem.

Czym jest zmienna?

Zmienna to pojęcie w programowaniu oznaczające **nazwę, którą przypisujemy pewnej wartości w pamięci komputera**. W programowaniu, **zmienne są używane do przechowywania i manipulowania danymi**. Wartość przechowywana przez zmienną może się zmieniać w trakcie wykonywania programu.

Aby zdefiniować zmienną, musimy określić jej nazwę i typ danych, który będzie przechowywany w zmiennej, chociaż nie zawsze jest to reguła, niektóre języki programowania nie wymagają narzucenia typu danych.

Nazwa zmiennej musi być unikalna i zgodna z zasadami składni języka programowania, w którym piszemy nasz program. Typ danych określa, jakiego rodzaju wartości może przechowywać zmienna, np. liczby całkowite, zmiennoprzecinkowe, tekstowe, wartości logiczne (`true`/`false`) itp.

Zmienne pozwalają programistom na tworzenie programów, które reagują na zmieniające się warunki i dane wejściowe. Przykładowo, w prostym programie kalkulatora zmienna może być użyta do przechowywania wprowadzonych przez użytkownika liczb, a następnie przeprowadzenie obliczeń na tych wartościach.

W programowaniu, wartości przypisane do zmiennych mogą być zmieniane w trakcie wykonywania programu, co czyni zmienne bardzo elastycznymi narzędziami programistycznymi. Zmienne pozwalają programistom na tworzenie dynamicznych aplikacji, które reagują na różne warunki i zmieniające się dane.

Przykładowo, w JavaScript, zmienne deklarowane są za pomocą słowa kluczowego `var`, `let` lub `const`.

Na przykład, można zadeklarować zmienną `x` i przypisać jej wartość `5`:

Przykład w JavaScript:

```
var x = 5;
```

W tym przykładzie, słowo kluczowe `var` oznacza, że tworzona jest zmienna `x`. Następnie, operator przypisania `=` przypisuje wartość `5` do tej zmiennej.

Zmienna może być przez Ciebie rozumiana jako bardzo podstawowy element wytworzonego programu. Przekładając to na analogię z życia, zmienna w programie komputerowym jest jak wkręt (śrubka) w jeżdżącym samochodzie.

Tłumaczenie dla dziecka

Pomyśl o zmiennych w programowaniu jak o pudełkach, w których można przechowywać różne rzeczy. Zmienne to miejsca w pamięci komputera, gdzie możemy zapisać dane i później z nich korzystać.

Na przykład, jeśli chcesz zapisać swoje imię w programie, możesz utworzyć zmienną o nazwie `imie` i przypisać do niej swoje imię. Gdy zmienna `imie` zostanie utworzona, komputer stworzy miejsce w pamięci, w którym zostanie zapisane Twoje imię. Później, jeśli będziesz potrzebował/-a użyć swojego imienia w programie, możesz odwołać się do zmiennej `imie` i skorzystać z zapisanego tam imienia.

Zmienne są bardzo przydatne w programowaniu, ponieważ pozwalają nam na przechowywanie informacji i wykorzystywanie ich w różnych miejscach w programie. Tak jak z pudełkami, możesz przechowywać różne rzeczy w różnych pudełkach i łatwiej jest odwołać się do konkretnego pudełka, gdy potrzebujesz jego zawartości.

Czym są typy danych w językach programowania?

Zmienne mogą narzucać typ wartości, jaki może być przypisany do danej zmiennej. Najprostsze typy danych to np. typ liczbowy, typ tekstowy, typ logiczny (`true`/`false`).

Typy danych w językach programowania określają rodzaj danych, jakie mogą być przechowywane w zmiennych. W zależności od języka programowania, typy danych mogą mieć różne nazwy i składnie, ale ogólnie można je podzielić na kilka kategorii:

1. **Typy proste:** Typy proste to podstawowe typy danych, które służą do przechowywania

pojedynczych wartości, takich jak liczby całkowite, zmiennoprzecinkowe, znaki, wartości logiczne (`prawda`/`fałsz`) i wartości `null` (`null` czyli brak wartości - nie ma, pusto, ale napisaliśmy o tym wprost). Typy proste są najczęściej używane w językach programowania i są zazwyczaj wbudowane w język.

2. **Typy złożone:** Typy złożone to typy danych, które służą do przechowywania złożonych struktur danych, takich jak tablice, listy, kolejki, czy drzewa (spokojnie, na pewnym etapie nauki będziesz mieć już dobre zrozumienie tego, co to jest i po co one są). Typy te pozwalają na przechowywanie większej liczby wartości w jednej zmiennej. Poruszymy sobie niedługo niektóre z nich.
3. **Typy wyliczeniowe:** Typy wyliczeniowe to typy danych, które służą do definiowania zestawu stałych wartości, takich jak dni tygodnia lub miesiąca. Z góry wiadomo, jakie 7 dni jest w tygodniu, czyli możesz narzucić, że zmienna może przyjmować tylko wartości: `poniedziałek`, `wtorek`... `niedziela`. Nie można wtedy do takiej zmiennej przypisać wartości *"jabłko"*, bo nie jest to już dzień tygodnia. Typy wyliczeniowe są przydatne, gdy chcemy ograniczyć zakres wartości, jakie mogą być przypisane do zmiennej.
4. **Typy referencyjne:** Typy referencyjne to typy danych, które służą do przechowywania referencji do obiektów w pamięci komputera. Typy te są używane w językach programowania obiektowego, takich jak Java czy C#, gdzie obiekty są tworzone i przekazywane między metodami. Czym są te obiekty? Wyjaśnimy poniżej.

W językach programowania typy danych określają, jakie operacje mogą być wykonywane na zmiennych oraz jakie wartości mogą być przypisane do tych zmiennych. Dlatego właściwe określenie typu danych jest ważne, aby zapewnić poprawne działanie programu i uniknąć błędów, wynikających np. z tego, że oczekujesz gruszki, a dostałeś jabłko.

Tłumaczenie dla dziecka

Typy danych w programowaniu to rodzaje informacji, jakie możemy przetwarzać na komputerze. Podobnie jak ludzie używają różnych słów do wyrażenia różnych rzeczy, tak w programowaniu mamy różne typy danych do przetwarzania różnych rodzajów informacji.

Na przykład, jeśli chcesz zapisać swoje imię w programie, musisz wybrać typ danych, który może przechowywać napisy (tekst). Jeśli chcesz zapisać swój wiek, musisz wybrać typ danych, który może przechowywać liczby.

W programowaniu, mamy wiele różnych typów danych, takich jak liczby całkowite, liczby zmiennoprzecinkowe (które zawierają miejsca dziesiętne), napisy (tekst), a także specjalne typy danych, takie jak wartości logiczne (`prawda`/`fałsz`).

To jest tak, jakbyśmy mieli różne pudła, do których wkładamy różne rzeczy. Do jednego pudła wkładamy książki, do drugiego zabawki, a do trzeciego ubrania. W programowaniu mamy różne *"pudła"* do przechowywania różnych typów danych.

Czym są funkcje i metody w językach programowania?

Funkcje i metody są reużywalnymi częściami programu, które możesz wykorzystywać wielokrotnie, gdzie masz taką potrzebę. Funkcje oraz metody można traktować jak oddzielne, niezależne programy, które można wywołać z głównego programu w celu wykonania określonego zadania.

Przykładem użycia funkcji w życiu może być np. korzystanie z kalkulatora. Kalkulator ma wiele różnych

funkcji, takich jak dodawanie, odejmowanie, mnożenie i dzielenie. Każda z tych funkcji jest wywoływana przez użytkownika, który podaje konkretne wartości liczbowe jako parametry.

Na przykład, gdy użytkownik chce dodać dwie liczby, wprowadza je do kalkulatora, a następnie wywołuje funkcję dodawania, która zwraca wynik dodawania tych dwóch liczb i wyświetla go na ekranie. Podobnie, gdy użytkownik chce pomnożyć dwie liczby, wprowadza je do kalkulatora i wywołuje funkcję mnożenia, która zwraca wynik mnożenia tych dwóch liczb i wyświetla go na ekranie.

Kalkulator jest dobrym przykładem użycia funkcji, ponieważ każda funkcja ma określony cel i wykonuje określone zadanie, a cały kalkulator składa się z wielu różnych funkcji, które są wywoływane w zależności od potrzeby użytkownika.

Tłumaczenie dla dziecka

Funkcje i metody w programowaniu to specjalne *"narzędzia"*, które pozwalają programistom na tworzenie kodu, który może być używany wielokrotnie.

Można to porównać do narzędzi w warsztacie. Na przykład, gdy Twój tata chce zbudować domek dla Ciebie, może użyć narzędzi, takich jak młotek i wkrętarka, aby zrobić różne rzeczy, takie jak wbicie gwoździ i przykręcenie desek. Funkcje i metody w programowaniu są podobne do tych narzędzi, ponieważ pozwalają na wielokrotne wykonywanie określonych czynności w kodzie.

Funkcje i metody są tworzone przez programistów i zawierają instrukcje, które określają, co komputer ma zrobić. Na przykład, może istnieć funkcja o nazwie *"dodaj"*, która dodaje dwa liczby razem. Programista może napisać kod dla funkcji *"dodaj"*, a następnie użyć jej wielokrotnie w programie, aby dodać różne pary liczb.

Gdy korzystasz z młotka, ten młotek został już raz przez kogoś stworzony (napisany), później go już tylko używasz. Gdy piszesz swoją funkcję, to jest trochę tak, jakbyś tworzył/-a swój własny młotek.

Funkcje i metody pozwalają na łatwe zarządzanie kodem i zmniejszenie ilości powtarzającego się kodu. Dzięki temu programiści mogą tworzyć bardziej skomplikowane programy i aplikacje, które są łatwiejsze w utrzymaniu i modyfikowaniu.

Czym są instrukcje warunkowe w językach programowania?

Wchodzimy na troszkę wyższy poziom skomplikowania. Omówmy kolejny element, który jest nieodzowną częścią powstających programów komputerowych.

Instrukcje warunkowe to konstrukcje w językach programowania, które pozwalają na wykonywanie różnych fragmentów kodu w zależności od spełnienia określonego warunku. Przypomnij sobie zdanie z początku tej książki: *"Jeżeli masz skończone 18 lat, możesz kupić alkohol, jeżeli nie masz to sorry!"*

Instrukcje warunkowe umożliwiają programistom tworzenie programów, które reagują na różne sytuacje i dane wejściowe.

W większości języków programowania istnieją dwie podstawowe formy instrukcji warunkowych:

Instrukcja IF

Instrukcja `if` to instrukcja, która pozwala na wykonanie jednego **bloku kodu, jeśli warunek jest spełniony**, lub innego bloku kodu, jeśli warunek nie jest spełniony. Warunek to wyrażenie logiczne, które zwraca wartość `true` lub `false`. Na przykład, spójrz na poniższy przykład w programie do zamawiania pizzy:

Przykład w Python:

```
if rozmiar_pizzy == "duża":
    cena_pizzy = 30
else:
    cena_pizzy = 20
```

Przykład sprawdza, czy wybrany rozmiar pizzy to *"duża"* i ustala odpowiednią cenę, w zależności od wyniku sprawdzenia.

Instrukcja switch/case

Instrukcja `switch/case` to instrukcja, która umożliwia **wykonywanie różnych bloków kodu w zależności od wartości zmiennej**. Instrukcja ta jest dostępna w niektórych językach programowania, takich jak C++, Java i PHP. Na przykład, spójrz na poniższy przykład w programie do zamawiania pizzy:

Przykład w Java:

```
switch (rodzaj_pizzy) {
    case "margherita":
        // kod dla pizzy margherita
        break;
    case "hawajska":
        // kod dla pizzy hawajskiej
        break;
    default:
        // kod dla innych rodzajów pizzy, które nie zostały wymienione wyżej
        break;
}
```

Przykład sprawdza wybrany rodzaj pizzy i wykonuje odpowiednią operację, w zależności od wartości zmiennej `rodzaj_pizzy`.

Tłumaczenie dla dziecka

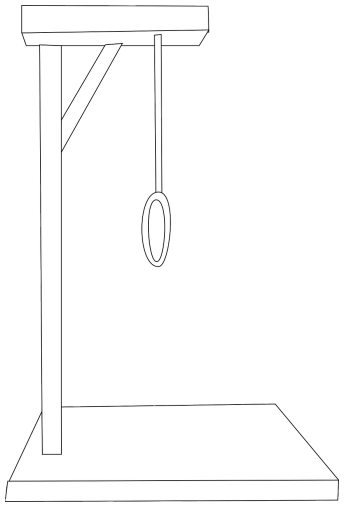
Pomyśl o instrukcjach warunkowych w programowaniu jak o instrukcjach *"jeśli - to"*. Możesz je wykorzystać, gdy chcesz, aby Twój komputer podjął decyzję na podstawie pewnych warunków.

Na przykład, jeśli ktoś pyta Cię *"czy dzisiaj pada deszcz?"*, to Ty patrzysz przez okno, żeby sprawdzić. W programowaniu możesz napisać kod, który mówi *"jeśli dzisiaj pada deszcz, to zabierz ze sobą parasol"*. To oznacza, że jeśli warunek (czyli *pada deszcz*) jest prawdziwy, to wykonaj instrukcję (*zabierz parasol*).

Możesz też zastosować instrukcje warunkowe do gier komputerowych. Na przykład, jeśli gracz zbliża się do przeszkody, to kod mówi, aby gracz przeskoczył ją, ale jeśli gracz nie zbliża się do przeszkody, to nie ma potrzeby skakania.

W ten sposób, instrukcje warunkowe w programowaniu pozwalają na podejmowanie decyzji przez komputer, co bardzo mocno upraszcza życie w wielu sytuacjach.

Czym w programowaniu są pętle?



Obraz 15. Źródło: https://pixabay.com

Jeżeli po dotarciu do tego miejsca naszej książki na słowo *"pętla"* nie pojawia Ci się w głowie powyższy obrazek, to znaczy, że jest dobrze. ☺ Więc pora przejść do konstrukcji, która w językach programowania służy do tego, żeby *"się nie powtarzać"*.

W programowaniu pętla to konstrukcja, która pozwala na **powtarzanie pewnej części kodu wielokrotnie**, do momentu osiągnięcia określonego warunku. Pętle są używane w programowaniu w celu zautomatyzowania powtarzających się zadań i zapewnienia bardziej efektywnego kodu.

W większości języków programowania istnieją trzy podstawowe rodzaje pętli (`for`, `while`, `do-while`).

Przypominamy, że w ramach tej książki będą pojawiały się fragmenty kodu, które nie będą na tym etapie zrozumiane przez Ciebie w 100% i to jest normalne. Szczegółowe wyjaśnienia poruszanych w tej książce zagadnień będą w kursie Zajavka. Ta książka ma Ci tylko zajawić temat i dać ogólne poczucie, że wiesz, w którym kościele dzwonią dzwony.

Dlatego nie oczekuj od siebie, że po przeczytaniu tej książki będziesz wszystko pamiętać i rozumieć. To przyjdzie z czasem.

Pętla for

Pętla `for` **wykonuje się określoną ilość razy**, zdefiniowaną przez programistę. Programista określa początkową wartość, końcową wartość oraz krok pętli, który jest dodawany do wartości początkowej w każdej iteracji.

Przykład w Java:

```
// Przykład: wypisanie liczb od 1 do 5 przy użyciu pętli for
for (int i = 1; i <= 5; i++) {
    System.out.println(i);
}
```

Wynik:

```
1
2
3
4
5
```

Pętla `for` jest często stosowana w Javie do wykonywania określonej liczby powtórzeń. W tym przykładzie, pętla `for` wykonuje się pięć razy, ponieważ zmienna `i` zaczyna od `1` i kończy na `5`, z każdą iteracją (wykonaniem, powtórzeniem) pętli zwiększamy wartość zmiennej `i` o jeden (zapis `i++` jest rozumiany jako zwiększenie wartości zmiennej `i` o 1). Każda iteracja pętli wypisuje wartość zmiennej `i` na ekran.

Pętla while

Pętla `while` wykonuje się, **dopóki określony warunek jest spełniony**. Programista musi zapewnić, że wewnętrzny warunek będzie miał wartość `true` lub `false`, w zależności od potrzeby, aby pętla działała prawidłowo.

Przykład w Java:

```
// Przykład: wypisanie liczb od 1 do 5 przy użyciu pętli while
int i = 1;
while (i <= 5) {
    System.out.println(i);
    i++;
}
```

Wynik:

```
1
2
3
4
5
```

Pętla `while` wykonuje się, dopóki warunek jest prawdziwy (czyli `i` jest mniejsze lub równe `5`). W tym przykładzie pętla while wypisuje liczby od `1` do `5`, ponieważ zmienna `i` zaczyna od `1`, a każda iteracja wypisuje wartość zmiennej `i` na ekran i zwiększa wartość `i` o `1`.

Pętla do-while

Pętla `do-while` jest podobna do pętli `while`, z tą różnicą, że **warunek jest sprawdzany na końcu każdej**

iteracji, a nie na początku. Oznacza to, że pętla `do-while` wykonuje się przynajmniej raz, niezależnie od tego, czy warunek jest spełniony, czy też nie.

Przykład w Java:

```
// Przykład: wypisanie liczb od 1 do 5 przy użyciu pętli do-while
int i = 1;
do {
    System.out.println(i);
    i++;
} while (i <= 5);
```

Wynik:

```
1
2
3
4
5
```

Pętla `do-while` jest podobna do pętli `while`, ale różni się tym, że warunek sprawdzany jest po wykonaniu instrukcji w pętli (a w pętli `while` warunek ten był sprawdzany przed jakimkolwiek wykonaniem pętli). W tym przykładzie, pętla `do-while` wypisuje liczby od `1` do `5`, ponieważ instrukcje w pętli są wykonane najpierw, a warunek sprawdzany jest po każdej iteracji. Pętla `do-while` jest przydatna w sytuacjach, gdy musimy wykonać pewną operację co najmniej raz, niezależnie od warunku.

Sklejamy do kupy

Pętle są bardzo ważne w programowaniu, ponieważ pozwalają programistom na powtarzanie operacji na różnych wartościach lub w różnych warunkach. Pętle pozwalają na bardziej czytelny, skrócony kod i zapewniają programistom elastyczność i kontrolę nad procesem wykonywania programu.

Pętle są również bardzo przydatne w programowaniu i pozwalają programistom na powtarzanie określonych operacji dla wielu wartości. Na przykład, pętle mogą być wykorzystywane w celu wykonania określonej operacji dla każdego elementu w liście lub tablicy (wyobraź sobie ciąg cyfr *1,2,3,4,5* - na tym etapie możesz w wielkim uproszczeniu rozumieć, że to jest lista lub tablica).

Przykładowo, załóżmy, że mamy listę liczb całkowitych i chcemy wyświetlić każdą z tych liczb pomnożoną przez 2. Możemy to zrobić za pomocą pętli `for` w języku Python:

Przykład w Python:

```
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    multiplied = num * 2
print(multiplied)
```

W tym przypadku, pętla `for` wykonuje się dla każdego elementu w liście `numbers`, a każda wartość jest mnożona przez `2` i wyświetlana na ekranie.

Innym przykładem zastosowania pętli może być obliczenie sumy liczb w zakresie od `1` do `10`. W języku

Java, możemy użyć pętli `for` (możemy też innych, po prostu ten przykład pokazuje pętlę `for`), aby obliczyć tę sumę:

Przykład w Java:

```
int sum = 0;
for (int i = 1; i <= 10; i++) {
    sum += i;
}
System.out.println("Suma liczb od 1 do 10 wynosi: " + sum);
```

W tym przypadku, pętla `for` wykonuje się od `1` do `10`, a każda wartość jest dodawana do zmiennej `sum`. Symbol `+=` oznacza *"zwiększ sumę `sum` o wartość zmiennej `i`"*. *Czyli z każdą iteracją pętli, wartość sum będzie zwiększana o `1`*, gdyż wartość `i` również jest zwiększana o `1`. Na końcu, wartość sumy jest wyświetlana na ekranie.

Tłumaczenie dla dziecka

Pomyśl o pętli w programowaniu jak o powtarzalnej zabawie. Pomyśl, że masz klocki i chcesz zbudować wieżę, ale nie chcesz budować jej tylko raz. Chcesz, aby Twoja wieża rosła i rosła, aż osiągnie niebo! Właśnie w tym momencie pętla przychodzi z pomocą.

Pętla to taka jakby maszyna, która pozwala Ci budować swoją wieżę klocków raz za razem bez konieczności zaczynania od nowa. Dzięki pętli możesz napisać kod, który mówi, aby Twój komputer budował Twoją wieżę tak wiele razy, jak chcesz.

Na przykład, gdybyś chciał zbudować wieżę z 10 klocków, możesz napisać kod, który mówi *"weź klocek, dodaj go do wieży, powtórz 9 razy"*. Komputer zrozumie, że ma powtórzyć tę instrukcję dziesięć razy i zbudować wieżę.

Tak więc, pętla w programowaniu to coś, co pomaga programiście wykonywać pewne instrukcje wielokrotnie, bez konieczności pisania tych instrukcji każdorazowo ręcznie.

OOP (Object Oriented Programming)

Wchodzimy w temat, który w kursie zajavka będzie wałkowany przez dobre kilka godzin. Nie spodziewaj się zatem, że poniższy opis da Ci pełne zrozumienie zagadnienia. Jedynie drapiemy ten temat po powierzchni.

Co to jest?

Programowanie obiektowe (**OOP** - **O**bject **O**riented **P**rogramming - Programowanie Zorientowane Obiektowo) to podejście do programowania, w którym programy są tworzone przez łączenie danych i funkcji w jednostkę zwane obiektem. **Obiekt to instancja** (konkretna realizacja) **klasy**, która zawiera określone dane i metody (funkcje), które mogą działać na tych danych.

Programowanie obiektowe można porównać do budowy domu. W tym porównaniu, klasa jest odpowiednikiem planu budynku, który określa cechy (wysokość, szerokość, ilość okien), które będą widoczne na wszystkich domach zbudowanych według tego samego planu. Klasa zawiera opis cech i zachowań, które są charakterystyczne dla obiektów utworzonych z tej klasy.

Obiekty reprezentują konkretne instancje klasy, tak jak domy reprezentują konkretne realizacje planu budynku. Każdy z tych domów ma swoje własne unikalne cechy, takie jak kolor elewacji, rozmiar czy układ pokojów. Podobnie obiekty mają unikalne właściwości, takie jak wartości zmiennych, które są zdefiniowane w klasie, a także stan wewnętrzny, w którym znajdują się podczas wykonywania kodu.

Metody to odpowiednik instrukcji, które programują zachowanie domu. Na przykład, metoda otwierająca okno powoduje otwarcie okna, a metoda włączająca światło powoduje, że światło zostanie włączone. Podobnie metody programowania obiektowego definiują zachowanie obiektów, takie jak działania, które można wykonać na obiekcie lub operacje, które można na nim przeprowadzić.

No i super. Rozejrzyj się dookoła. Ile widzisz obiektów? Mnóstwo! Krzesło, stół, monitor, klawiatura - to wszystko są obiekty. Obiekty mają zdefiniowane swoje cechy (atrybuty) i zachowania (metody). Programiści piszący programy obiektowe często patrzą na świat w przez pryzmat obiektów.

Tłumaczenie dla dziecka

W programowaniu **OOP**, coś takiego jak *"klasa"* to taki sam rodzaj rzeczy jak *"książka"* czy *"samochód"*. Każda książka ma swoje specjalne cechy - tytuł, autor, ilustracje - a samochód ma swoje cechy - kolor, rozmiar, liczba drzwi. Podobnie każda klasa ma swoje własne specjalne cechy, na przykład *"pies"* może mieć cechy takie jak kolor sierści, wielkość łapy, i wiek.

Załóżmy, że piszemy program, który ma pomóc nam w zabawie w układanie puzzli. Zamiast ręcznie układać puzzle, komputer może pomóc nam w rozwiązaniu zadania. W programowaniu **OOP**, używamy klas, aby opisać poszczególne elementy puzzli i sposoby, w jakie można je układać. Klasa może mieć swoje własne cechy, takie jak kształt i kolor, a także specjalne funkcje, takie jak *"porównaj ten kawałek z tym innym"*, aby upewnić się, że pasują do siebie.

Programowanie OOP pomaga programistom tworzyć bardziej złożone programy, które są łatwiejsze do zrozumienia, modyfikowania i utrzymywania. Dzięki OOP, można tworzyć aplikacje, które są bardziej elastyczne i łatwiej dostosowywane do zmieniających się wymagań.

Jakie są podstawy programowania obiektowego?

Jak każda koncepcja, tak i OOP jest związane z pewnymi założeniami. Poniżej przedstawimy kilka z nich:

1. **Koncepcja obiektów:** programowanie obiektowe jest oparte na koncepcji obiektów, które są abstrakcyjnymi pojęciami reprezentującymi rzeczywiste obiekty. Obiekty składają się z atrybutów (właściwości) i metod (funkcji), które umożliwiają interakcję z nimi.
2. **Klasa:** klasa jest szablonem definiującym cechy i zachowania obiektów. Klasa definiuje atrybuty i metody, które są dziedziczone przez obiekty utworzone na jej podstawie.
3. **Dziedziczenie:** dziedziczenie pozwala na tworzenie klas pochodnych, które dziedziczą atrybuty i metody po klasie bazowej. Klasa pochodna może rozszerzyć lub nadpisać metody klasy bazowej. Trochę tak jak dziecko może dziedziczyć cechy i zachowania rodzica.

Kolejne dwa założenia podejścia obiektowego są już skomplikowane i wyjaśniamy je dogłębnie na ścieżce Zajavka. Jednakże, żeby ta książka była kompletna, powiedzmy sobie o nich również tutaj:

4. **Polimorfizm:** polimorfizm umożliwia zastosowanie różnych implementacji metody w zależności od

typu obiektu, który ją wywołuje. W programowaniu obiektowym, polimorfizm jest często osiągany poprzez przesłanianie metod w klasach pochodnych.

5. **Enkapsulacja:** enkapsulacja polega na ukrywaniu atrybutów i metod obiektu przed światem zewnętrznym. Obiekty zapewniają interfejs do swojego zachowania, który pozwala na ich interakcję z innymi obiektami bez potrzeby poznawania wewnętrznej implementacji.

Przykładem języka programowania, który stosuje programowanie obiektowe, jest JavaScript (Java również, ale dla odmiany podamy przykład w JavaScript). W JavaScript, obiekty tworzone są poprzez słowo kluczowe `new` i konstruktor klasy. Przykładowa klasa w JavaScript może wyglądać następująco:

Przykład w JavaScript:

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    sayHello() {
        console.log(
            "Hello, my name is "
            + this.name
            + " and I'm "
            + this.age
            + " years old."
        );
    }
}
```

W tym przykładzie, klasa `Person` definiuje atrybuty `name` i `age` oraz metodę `sayHello()`. Obiekt klasy `Person` może być utworzony za pomocą konstruktora:

Przykład w JavaScript:

```
let person = new Person("John", 30);
```

W tym przypadku, utworzony został obiekt `person` (z małej litery) na podstawie klasy `Person` z wartościami `name` równą `John` i `age` równą `30`. Można wywołać metodę `sayHello()` na obiekcie `person`:

Wywołanie metody `sayHello()`:

```
person.sayHello();
```

Co w efekcie wyświetli na ekranie:

```
Hello, my name is John and I'm 30 years old.
```

Jaka jest różnica pomiędzy klasą a obiektem?

Klasa i obiekt to dwa pojęcia związane z programowaniem obiektowym. Klasa jest szablonem, wzorcem

lub definicją, z której tworzone są obiekty, podczas gdy obiekt jest egzemplarzem klasy.

Klasa definiuje atrybuty i metody, które są dziedziczone przez obiekty utworzone na jej podstawie. Klasa określa cechy obiektów, takie jak atrybuty, metody, właściwości i zachowania, ale nie posiada konkretnych wartości tych atrybutów i metod.

Obiekt jest instancją klasy i posiada konkretne wartości atrybutów i metod określonych w klasie. Obiekt to abstrakcyjna reprezentacja rzeczywistego obiektu, który ma swoje własne wartości atrybutów i metody. Każdy obiekt jest niezależny od innych obiektów utworzonych na podstawie tej samej klasy i może mieć różne wartości atrybutów i metody.

Na przykład, można utworzyć klasę Person i zdefiniować w niej atrybuty i metody. Następnie, można utworzyć dwa różne obiekty klasy Person o różnych wartościach atrybutów:

Przykład w JavaScript:

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    sayHello() {
        console.log(
            "Hello, my name is "
            + this.name
            + " and I'm "
            + this.age
            + " years old."
        );
    }
}
```

Stwórzmy teraz dwa obiekty klasy `Person`.

Przykład w JavaScript:

```
let person1 = new Person("John", 30);
let person2 = new Person("Jane", 25);
```

Następnie dla tych dwóch obiektów `person1` i `person2` wywołajmy metody `sayHello()`.

Wynik wywołania:

```
person1.sayHello(); // "Hello, my name is John and I'm 30 years old."
person2.sayHello(); // "Hello, my name is Jane and I'm 25 years old."
```

W tym przykładzie, klasa `Person` definiuje atrybuty `name` i `age` oraz metodę `sayHello()`. Następnie, tworzone są dwa obiekty klasy `Person` o różnych wartościach atrybutów i przypisane do dwóch różnych zmiennych (`person1` i `person2`). Wywołanie metody `sayHello()` na każdym z obiektów wyświetli różne wartości zdefiniowane w każdym z obiektów.

Po ludzku, co my tu właśnie zrobiliśmy? Stworzyliśmy w programie dwie osoby (John i Jane),

określiliśmy, ile mają lat, a następnie kazaliśmy im się przywitać, poprzez wywołanie metody `sayHello()`.

Struktury danych

Przechodzimy do kolejnego zagadnienia, które w programowaniu jest bardzo istotne. Jest ono związane ze sposobem przechowywania danych. Czym są te struktury?

Co to i po co to?

W programowaniu struktury danych to złożone obiekty, które pozwalają na przechowywanie i organizowanie dużej ilości powiązanych ze sobą danych. Struktury danych umożliwiają tworzenie bardziej skomplikowanych programów i algorytmów poprzez przechowywanie danych w bardziej zorganizowany sposób.

Struktury danych składają się z jednego lub więcej typów danych, takich jak liczby, napisy lub wartości logiczne. Mogą też zawierać inne struktury danych, co umożliwia tworzenie bardziej złożonych hierarchii.

Przykładem struktury danych może być baza danych zawierająca informacje o pracownikach w firmie. Struktura ta może zawierać kilka pól, takich jak imię, nazwisko, adres, numer telefonu i stanowisko. Dane te są ze sobą powiązane i pozwalają na łatwe wyszukiwanie i zarządzanie informacjami o pracownikach w firmie.

Innym przykładem struktury danych może być lista kontaktów w telefonie komórkowym. Struktura ta zawiera różne pola, takie jak imię, nazwisko, numer telefonu, adres e-mail itp. Dane te są przechowywane w sposób zorganizowany i pozwalają na łatwe odnajdywanie i zarządzanie kontaktami.

W programowaniu struktury danych są bardzo przydatne, ponieważ pozwalają na przechowywanie i organizowanie dużej ilości danych w sposób zrozumiały dla komputera. Dzięki temu programista może łatwo odwoływać się do danych, manipulować nimi i tworzyć bardziej skomplikowane programy i algorytmy.

Tłumaczenie dla dziecka

Pomyśl o strukturach danych w programowaniu jak o skrzynkach z przegródkami, w których można przechowywać różne rzeczy. Struktury danych pozwalają na przechowywanie wielu informacji w jednym miejscu i łatwiejsze zarządzanie nimi.

Na przykład, jeśli chcesz przechowywać informacje o swoich ulubionych kolorach, mógłbyś stworzyć strukturę danych o nazwie *"ulubione kolory"*. Wewnątrz struktury danych *"ulubione kolory"* można utworzyć kilka przegródek (tzw. elementów) na różne kolory, takie jak *"zielony"*, *"niebieski"* i *"czerwony"*.

Dzięki strukturze danych, możesz łatwo przechowywać wiele różnych kolorów w jednym miejscu i odwoływać się do nich, kiedy ich potrzebujesz. Możesz również dodawać nowe kolory lub usuwać stare kolory ze struktury danych.

Struktury danych są bardzo przydatne w programowaniu, ponieważ pozwalają na przechowywanie wielu informacji w jednym miejscu i ułatwiają zarządzanie nimi. Tak jak skrzynki z przegródkami, struktury danych pozwalają na uporządkowanie informacji i łatwe ich odnalezienie, kiedy są potrzebne.

Czym w programowaniu są tablice?

Nawet jeśli wszystkiego nie umiesz i żadna facetka Cię nigdzie nie wzięła, to zaraz wszystko stanie się jasne. Zacznijmy od jednej z najbardziej popularnych struktur danych, czyli tablic.

Tablice są jednym z podstawowych typów danych w programowaniu. Służą do przechowywania wielu wartości tego samego typu w jednej zmiennej. Tablice pozwalają na łatwe przechowywanie i operowanie na zbiorach danych, co czyni je bardzo użytecznymi w programowaniu.

Tablica składa się z wielu elementów, które są ułożone w kolejności. Każdy element ma przypisany indeks, który określa jego pozycję w tablicy. Indeksy zaczynają się od 0 dla pierwszego elementu, a kończą na długości tablicy minus 1 dla ostatniego elementu.

Tablice mogą przechowywać różne typy danych, takie jak liczby, tekst, wartości logiczne, a nawet inne tablice. Elementy tablicy mogą być modyfikowane, dodawane lub usuwane za pomocą indeksów.

Oto kilka podstawowych operacji, które można wykonywać na tablicach:

1. **Tworzenie tablicy:** tablica może być utworzona za pomocą znaków `[]` w językach takich jak Python lub za pomocą słówka `new` w językach takich jak Java.
2. **Dodawanie elementów do tablicy:** elementy mogą być dodawane na końcu tablicy za pomocą funkcji, takich jak `append()` w Pythonie lub `push()` w Javie.
3. **Usuwanie elementów z tablicy:** elementy mogą być usuwane z tablicy za pomocą funkcji, takich jak pop w Pythonie lub remove w Javie.
4. **Odczytywanie elementów tablicy:** elementy tablicy mogą być odczytywane za pomocą ich indeksów.
5. **Modyfikowanie elementów tablicy:** elementy tablicy mogą być modyfikowane poprzez przypisanie nowej wartości do odpowiedniego indeksu.

Tablice są niezbędne do przechowywania i manipulowania danymi w programowaniu. Dzięki nim programiści mogą łatwo przechowywać dane w jednym uporządkowanym miejscu.

Jednym z przykładów operacji na tablicach w języku Java może być sortowanie tablicy. Swoją drogą sposobów na posortowanie elementów w tablicy jest bardzo dużo, zobacz na Wikipedii.

Przykład sortowania tablicy w Java:

```java
// Określenie zawartości tablicy
int[] tablica = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};

// Sortowanie tablicy
Arrays.sort(tablica);

// Wypisanie posortowanej tablicy
for (int liczba : tablica) {
    System.out.print(liczba + " ");
}
```

W tym przykładzie tworzymy tablicę liczb całkowitych o nazwie `tablica`, a następnie sortujemy ją za pomocą metody `sort()` w klasie `Arrays`. W końcu, wypisujemy posortowaną tablicę na ekran.

Wynik sortowania:

```
1 1 2 3 3 4 5 5 5 6 9
```

Tłumaczenie dla dziecka

Pomyśl o tablicach w programowaniu jak o skrzynkach na klocki, które pozwalają na przechowywanie wielu takich samych klocków w jednym miejscu. Tablice to zbiór elementów tego samego typu, które są ułożone w określonym porządku.

Na przykład, jeśli chcesz przechowywać kilka liczb, możesz utworzyć tablicę liczb całkowitych, która zawiera kilka elementów. Każdy element tablicy jest indeksowany, co oznacza, że można się do niego odwołać za pomocą jego numeru indeksu.

Przykładowo, tablica liczb całkowitych o rozmiarze 4 może zawierać liczby 2, 4, 6 i 8. Każda liczba zostanie przypisana do określonego miejsca w tablicy, a jej numer indeksu oznacza miejsce, w którym się znajduje. Czyli, liczba 2 jest w miejscu 0, liczba 4 jest w miejscu 1, liczba 6 jest w miejscu 2, a liczba 8 jest w miejscu 3.

Dzięki tablicom, możesz łatwo przechowywać wiele elementów w jednym miejscu i odwoływać się do nich, kiedy ich potrzebujesz. Tak jak skrzynki na klocki, tablice pozwalają na uporządkowanie elementów i łatwe ich odnalezienie, kiedy są potrzebne.

Czym w programowaniu są listy?

Obraz 16. Źródło: Luknij.net

W programowaniu lista to kolekcja elementów o różnych typach, które są uporządkowane w określonej kolejności. Listy są jednym z podstawowych typów danych i są często używane w programowaniu do przechowywania i manipulowania dużymi zbiorami danych.

Listy są elastyczne i pozwalają na dodawanie i usuwanie elementów w czasie działania programu, co sprawia, że są one bardzo przydatne w sytuacjach, gdy liczba elementów jest dynamiczna lub zmienia się w czasie. Każdy element na liście ma przypisany indeks, który określa jego pozycję w liście.

W języku Java, listy są definiowane za pomocą klasy `List`, a elementy na liście są dodawane za pomocą metody `add()`:

Przykład w Java:

```
List<String> my_list = new ArrayList<>();
my_list.add("jeden");
my_list.add("dwa");
my_list.add("trzy");
```

Listy są często używane w programowaniu do przechowywania i przetwarzania dużych ilości danych, takich jak wyniki pomiarów, dane użytkowników lub informacje o produktach. Listy pozwalają na łatwe wyszukiwanie, sortowanie i filtrowanie danych, co jest niezbędne w przypadku wielu zastosowań programistycznych.

Tłumaczenie dla dziecka

W programowaniu, lista to taki sam rodzaj rzeczy jak lista zakupów, która pomaga nam zapamiętać rzeczy, które chcemy kupić. Lista w programowaniu pomaga nam przechowywać różne elementy, takie jak liczby lub słowa, w jednym miejscu. Na liście możemy przechowywać wiele elementów i mieć do nich łatwy dostęp.

Każdy element na liście ma swój własny numer, nazywany indeksem. Na przykład, jeśli mamy listę składającą się z pięciu elementów, pierwszy element ma indeks 0, drugi element ma indeks 1, a ostatni element ma indeks 4. Możemy dodawać elementy do listy lub usuwać je z niej w dowolnym momencie. To sprawia, że lista jest bardzo elastyczna i przydatna w programowaniu.

Listy są często używane w programowaniu, ponieważ pozwalają nam na przechowywanie wielu elementów i manipulowanie nimi w łatwy sposób. Na przykład, jeśli chcemy wyświetlić wszystkie elementy na liście, możemy użyć pętli, która wyświetli każdy element na liście. To sprawia, że lista jest bardzo przydatna w wielu różnych rodzajach programów.

Czym różni się tablica od listy?

W programowaniu tablica i lista to dwa różne sposoby przechowywania danych. Tablica to struktura danych, która przechowuje elementy tego samego typu w kolejności, która jest określona przez indeksy. Indeks to liczba, która identyfikuje pozycję elementu w tablicy. Na przykład, pierwszy element w tablicy ma indeks 0, drugi element ma indeks 1, a trzeci element ma indeks 2, i tak dalej.

Lista to struktura danych, która przechowuje elementy w kolejności, którą określamy przy ich dodawaniu do listy. Elementy w liście mogą mieć różne typy, a każdy element ma swój własny indeks, który jest automatycznie nadawany przez listę. Na przykład, pierwszy element w liście ma indeks 0, drugi element ma indeks 1, a trzeci element ma indeks 2, i tak dalej.

Różnica między tablicą a listą polega na sposobie przechowywania i manipulowania danymi. W tablicy, rozmiar jest określony podczas tworzenia, a indeksy są wykorzystywane do dostępu do elementów w tablicy. W liście, rozmiar jest dynamicznie zmieniany, a elementy mogą być dodawane lub usuwane w dowolnym miejscu na liście. Listy są bardziej elastyczne i łatwiejsze w użyciu, ale tablice są szybsze i bardziej wydajne, gdy wymagana jest szybka obsługa danych.

Czym w programowaniu są drzewa?

Obraz 17. Źródło: https://kwejk.pl

Mam nadzieję, że jeszcze nie masz dość, bo przechodzimy teraz do drzew. Całe szczęście, drzewa w programowaniu, tak samo, jak w normalnym życiu mają liście.

W programowaniu, drzewo to struktura danych, która składa się z węzłów połączonych krawędziami w taki sposób, że każdy węzeł ma jeden węzeł nadrzędny, z wyjątkiem węzła korzenia, który nie ma węzła nadrzędnego. Drzewo może mieć wiele gałęzi, które rozgałęziają się od jednego węzła i prowadzą do innych węzłów.

Drzewa są bardzo przydatne w programowaniu, ponieważ pozwalają na hierarchiczne organizowanie danych i łatwe przeszukiwanie ich w określony sposób. Drzewa są często wykorzystywane w algorytmach wyszukiwania, sortowania i analizy danych, a także w systemach plików, bazach danych i strukturach sieciowych. Drzewa są również często reprezentowane w formie diagramów, które pozwalają na łatwe zrozumienie ich struktury i relacji między węzłami.

Tłumaczenie dla dziecka

W programowaniu, drzewo to taki sam rodzaj rzeczy jak drzewo w parku, ale zamiast liści, gałęzi i pnia, składa się z węzłów. Każdy węzeł w drzewie reprezentuje pewną wartość lub element, a drzewo składa się z połączonych ze sobą węzłów.

Drzewo jest zbudowane w sposób hierarchiczny, co oznacza, że węzły są ułożone w układzie, który przypomina gałęzie i korzenie drzewa. Każdy węzeł ma swojego *"rodzica"* lub *"ojca"*, poza pierwszym węzłem, który jest uważany za korzeń drzewa. Węzły, które pochodzą od tego samego ojca, są nazywane *"dziećmi"* lub *"potomkami"*. W ten sposób drzewo reprezentuje strukturę, w której jeden element jest powiązany z innymi elementami.

Drzewa są często używane w programowaniu, ponieważ pozwalają na przechowywanie i organizowanie danych w sposób hierarchiczny. Na przykład, drzewo może reprezentować strukturę folderów i plików na komputerze, gdzie każdy folder jest węzłem, a pliki są dziećmi tego węzła. Może też reprezentować hierarchię kategorii produktów w sklepie internetowym, gdzie każda kategoria jest

węzłem, a produkty są dziećmi tego węzła.

Dzięki drzewom, programiści mogą szybko wyszukiwać, sortować i manipulować dużymi ilościami danych w sposób skuteczny i wydajny.

Czym w programowaniu są słowniki?

Obraz 18. Źródło: https://kwejk.pl

W programowaniu, słownik to struktura danych, która przechowuje pary klucz-wartość. Każdy klucz na słowniku jest unikalny i jest przypisany do jednej wartości. Słowniki są często używane w programowaniu do przechowywania i manipulowania dużymi zbiorami danych, w których wartości są indeksowane przez unikalne klucze.

W słowniku, każda para klucz-wartość jest przechowywana w oddzielnym elemencie, który jest wykorzystywany do szybkiego dostępu do wartości, które są powiązane z konkretnym kluczem. Słowniki pozwalają programistom na szybkie wyszukiwanie, dodawanie i usuwanie elementów na podstawie klucza.

Słowniki są często reprezentowane w formie tabel, które składają się z dwóch kolumn: kluczy i wartości. W większości języków programowania, słowniki są reprezentowane jako typy danych wbudowane, które pozwalają na łatwe dodawanie, usuwanie i modyfikowanie elementów. Słowniki są bardzo elastyczne i pozwalają na przechowywanie elementów o różnych typach, takich jak liczby, ciągi znaków, obiekty i wiele innych.

Na przykład, w języku Python, słowniki są definiowane za pomocą nawiasów klamrowych i par klucz-wartość są oddzielone przecinkami:

Przykład w Python:

```
my_dict = {"klucz1": "wartosc1", "klucz2": 2, "klucz3": true}
```

W języku Java, słowniki są reprezentowane za pomocą klasy `Map`, która umożliwia dodawanie, usuwanie i modyfikowanie elementów na podstawie klucza:

Przykład w Java:

```
Map<String, Integer> my_dict = new HashMap<>();
my_dict.put("klucz1", 1);
my_dict.put("klucz2", 2);
my_dict.put("klucz3", 3);
```

Słowniki są bardzo przydatne w programowaniu, ponieważ pozwalają na łatwe przechowywanie i przetwarzanie dużej ilości danych, w których wartości są powiązane z określonymi kluczami. Słowniki są często wykorzystywane w algorytmach przetwarzania tekstu, systemach baz danych i innych aplikacjach programistycznych, które wymagają skutecznego przechowywania i manipulowania danymi.

Tłumaczenie dla dziecka

W programowaniu, słownik to taki sam rodzaj rzeczy jak słownik, który używamy do znalezienia definicji słowa. Słownik w programowaniu pomaga nam przypisać jednej wartości do innej wartości, podobnie jak definicja przypisuje słowo do jego znaczenia.

Słownik składa się z par klucz-wartość, gdzie klucz to unikalny identyfikator dla danej wartości, a wartość to konkretna wartość, która jest z nią powiązana. Na przykład, jeśli chcemy przypisać oceny za testy do imion uczniów, możemy użyć słownika, gdzie kluczem będzie imię ucznia, a wartością będzie ocena, którą uzyskał.

Słowniki są często używane w programowaniu, ponieważ pozwalają na przechowywanie i organizowanie danych w sposób łatwy do wyszukiwania i modyfikowania. Na przykład, możemy użyć słownika, aby przechowywać informacje o użytkownikach, takie jak ich adresy e-mail, numery telefonów i imiona. Możemy również użyć słownika, aby przechowywać informacje o produktach w sklepie internetowym, takie jak nazwy, ceny i zdjęcia.

Dzięki słownikom, programiści mogą szybko i łatwo przypisywać wartości do kluczy i wyszukiwać wartości po kluczu. To sprawia, że słowniki są bardzo przydatne w wielu różnych rodzajach programów.

Jakie są podstawowe zasady programowania?

Każda dziedzina (łyżwiarstwo figurowe, mechanika samochodowa czy wspinaczka) cechują się pewnymi zasadami, które zostały wypracowane przez ludzi na przestrzeni lat i których należy przestrzegać, żeby robić coś prawidłowo. Oczywiście można wspinać się na górę w pozycji do góry nogami, albo jeździć na łyżwach bez łyżew, ale po co, skoro grono ludzi, które robiło to już przed Tobą, wypracowało zestaw zasad i technik określających jak robić to dobrze.

W programowaniu jest tak samo! Tutaj też istnieją pewne zasady, które mają Ci pomóc pisać czytelne i zrozumiałe programy. Wymieńmy sobie pokrótce kilka podstawowych zasad, jakimi powinien kierować

się programista (zasad jest o wiele więcej i poruszamy je w trakcie całego kursu zajavka):

1. **Zasada DRY** (Don't Repeat Yourself): unikaj powtarzania kodu. Jeśli dany fragment kodu jest używany w kilku miejscach, lepiej utworzyć funkcję lub metodę, która będzie mogła być wykorzystywana wielokrotnie.
2. **Zasada KISS** (Keep It Simple, Stupid): twórz proste rozwiązania. Im prostszy kod, tym łatwiejsze będzie jego utrzymanie, modyfikacja i rozwijanie.
3. **Zasada YAGNI** (You Ain't Gonna Need It): nie dodawaj zbędnych funkcjonalności do kodu. Koncentruj się na tworzeniu tylko tych elementów, które są naprawdę potrzebne.
4. **Zasada TDD** (Test-Driven Development): twórz kod, który jest testowalny. TDD polega na pisaniu testów przed napisaniem kodu, co pomaga w tworzeniu bardziej niezawodnego i stabilnego kodu. O testach wspomnimy jeszcze za moment.
5. **Zasada nazewnictwa**: nazwij zmienne, funkcje i klasy w sposób zrozumiały dla innych programistów. Dobrze dobrane nazwy pomagają w łatwiejszym zrozumieniu kodu i zapobiegają pomyłkom.

Wspomniane zasady nie są jedynymi, z jakimi spotkasz się na swojej ścieżce programisty i tak jak wspomnieliśmy, na ścieżce zajavka poznasz tych zasad o wiele więcej. Przedstawione zasady pomagają w tworzeniu czytelnego, niezawodnego i elastycznego kodu, który jest łatwy w utrzymaniu i rozwijaniu.

Algorytmy

Pora na algorytmy! Czyli takie coś, z takim czymś, ale bez takiego czegoś! Gotowy/-a? To jedziemy.

Czym jest algorytm?

W programowaniu, algorytm to taki przepis na zrobienie ciastek. Algorytm to zestaw instrukcji, które określają, jak rozwiązać określony problem krok po kroku. Instrukcje te powinny być jasne i precyzyjne, aby można było je łatwo zrozumieć i zastosować.

Algorytmy są często używane w programowaniu, aby rozwiązywać różne problemy, takie jak sortowanie danych, wyszukiwanie informacji czy przetwarzanie obrazów. Algorytmy są ważne, ponieważ pomagają nam w tworzeniu bardziej wydajnych i skutecznych programów.

Na przykład, gdy piszemy program, który ma posortować listę liczb od najmniejszej do największej, musimy napisać algorytm, który określa, jak to zrobić. Algorytm może polegać na porównywaniu dwóch liczb i zamianie ich miejscami, jeśli jedna liczba jest większa niż druga. Algorytm ten powtarzany jest dla każdej pary liczb, aż do momentu, gdy lista zostanie posortowana.

Czy sztuczna inteligencja też jest algorytmem?

Nie, sztuczna inteligencja (AI) to ogólny termin, który obejmuje różne metody i technologie wykorzystywane do tworzenia systemów, które mogą naśladować ludzką inteligencję i podejmować decyzje na podstawie analizy danych. AI składa się z wielu elementów, w tym z algorytmów, które są jednym z narzędzi, które pomagają programistom stworzyć systemy sztucznej inteligencji.

Algorytmy są ważnymi elementami sztucznej inteligencji, ponieważ służą do przetwarzania danych i podejmowania decyzji. Na przykład, algorytmy uczenia maszynowego są wykorzystywane do

trenowania systemów sztucznej inteligencji, aby były w stanie analizować dane i uczyć się z nich, a algorytmy sieci neuronowych są wykorzystywane do przetwarzania obrazów i dźwięków.

Jednakże, sztuczna inteligencja to bardziej złożone pojęcie niż sam algorytm, ponieważ obejmuje wiele różnych dziedzin, takich jak uczenie maszynowe, przetwarzanie języka naturalnego i rozpoznawanie wzorców. Wszystkie te dziedziny są połączone w celu stworzenia systemów sztucznej inteligencji, które potrafią rozwiązywać skomplikowane problemy i podejmować decyzje na podstawie analizy danych.

Teraz wymienimy kilka podstawowych algorytmów

Algorytm sortowania bąbelkowego

Ten algorytm służy do sortowania tablicy liczb. Działanie algorytmu polega na porównywaniu sąsiednich elementów tablicy i zamianie ich kolejności, jeśli są ustawione w nieodpowiedniej kolejności. Algorytm przechodzi po tablicy wielokrotnie, aż wszystkie elementy zostaną ułożone w kolejności rosnącej.

Algorytm silnia

Ten algorytm służy do obliczania silni liczby. Silnia to iloczyn wszystkich liczb naturalnych od 1 do danej liczby. Algorytm silni polega na mnożeniu kolejnych liczb naturalnych, począwszy od 1, aż do danej liczby. Na przykład silnia liczby 5 wynosi *5 * 4 * 3 * 2 * 1 = 120*.

Na tej podstawie widać, że, algorytmy to zestawy instrukcji, które wykonują określone zadanie lub rozwiązują problem.

Omówiliśmy sobie te algorytmy w bardzo podstawowym zakresie. Spójrzmy na nie jednak trochę szerzej.

Głębsze spojrzenie na algorytmy

Algorytm sortowania bąbelkowego

Algorytm sortowania bąbelkowego to prosta metoda sortowania tablicy liczb, która działa poprzez porównywanie sąsiednich elementów i zamianę ich kolejności, jeśli są ustawione w nieodpowiedniej kolejności. Algorytm przechodzi po tablicy wielokrotnie, aż wszystkie elementy zostaną ułożone w kolejności rosnącej lub malejącej. Nazwa *"sortowanie bąbelkowe"* pochodzi od wizualnego efektu, jaki powstaje w trakcie sortowania - mniejsze elementy *"wypływają"* na początek tablicy, podobnie jak pęcherzyki powietrza w wodzie.

Algorytm sortowania bąbelkowego można zaimplementować w następujący sposób:

1. Przechodzimy po tablicy i porównujemy sąsiadujące elementy.
2. Jeśli element znajdujący się na prawo jest mniejszy niż element znajdujący się na lewo, zamieniamy ich kolejność.
3. Przechodzimy po tablicy wielokrotnie, aż nie dokonamy żadnej zamiany.

Oto przykładowy kod w języku Python, który implementuje algorytm sortowania bąbelkowego:

Przykład w Python:

```
def bubble_sort(arr):
    n = len(arr)
    # przeglądamy elementy tablicy
    for i in range(n):
        # ostatnie i elementów jest już posortowanych, pomijamy je
        for j in range(0, n-i-1):
            # zamieniamy elementy, jeśli są w złej kolejności
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

Wywołanie w Python:

```
# przykładowa lista
arr = [64, 34, 25, 12, 22, 11, 90]

# sortowanie listy
bubble_sort(arr)

# wyświetlenie posortowanej listy
print("Posortowana lista:")
for i in range(len(arr)):
    print("%d" %arr[i])
```

W powyższym kodzie zaimplementowana jest funkcja `bubble_sort()`, która przyjmuje jako argument tablicę liczb `arr` i sortuje ją w miejscu. Wewnętrzna pętla for iteruje po nieposortowanych elementach tablicy, a następnie porównuje sąsiednie elementy i zamienia je, jeśli są w złej kolejności. Zewnętrzna pętla for powtarza ten proces dla każdego elementu tablicy, aż do momentu, gdy cała tablica zostanie posortowana.

W przykładzie wykorzystana jest lista liczb do posortowania, ale algorytm sortowania bąbelkowego można zastosować do dowolnego typu danych, które można porównać i zamienić miejscami.

Podsumowując, algorytm sortowania bąbelkowego to prosty i łatwy do zaimplementowania sposób sortowania tablicy liczb. Algorytm działa poprzez porównywanie sąsiednich elementów i zamianę ich kolejności, jeśli są ustawione w nieodpowiedniej kolejności. Algorytm przechodzi po tablicy wielokrotnie, aż wszystkie elementy zostaną ułożone w kolejności rosnącej lub malejącej.

Ciekawostka: Chcesz zobaczyć ciekawszy sposób na sortowanie bąbelkowe? Obejrzyj ten film: Bubble sort with Hungarian, folk dance.

Algorytm silnia

Algorytm silnia to prosta metoda obliczania wartości silni liczby. Silnia to iloczyn wszystkich liczb naturalnych od 1 do danej liczby. Algorytm silni polega na mnożeniu kolejnych liczb naturalnych, począwszy od 1, aż do danej liczby.

Algorytm silni można zaimplementować w następujący sposób:

1. Ustalamy liczbę, której silnię chcemy obliczyć.
2. Tworzymy zmienną wynikową i przypisujemy do niej wartość 1.

3. Przechodzimy po kolejnych liczbach naturalnych od 1 do danej liczby i mnożymy je przez wartość zmiennej wynikowej.
4. Po osiągnięciu danej liczby, zwracamy wartość zmiennej wynikowej jako wynik.

Oto przykładowy kod w języku Python, który implementuje algorytm silni:

Przykład w Python:

```
def silnia(n):
    wynik = 1
    for i in range(1, n + 1):
        wynik = wynik * i
    return wynik
```

Powyższy kod oblicza silnię liczby `n` za pomocą pętli `for`. Algorytm przechodzi po kolejnych liczbach naturalnych od `1` do `n` i mnoży je przez wartość zmiennej wynikowej. Po osiągnięciu `n`, zwracana jest wartość zmiennej wynikowej jako wynik.

Algorytm silni może być również zaimplementowany rekurencyjnie.

Czym jest rekurencja?

Rekurencja to technika programowania polegająca na wywoływaniu funkcji przez samą siebie. Innymi słowy, funkcja rekurencyjna to funkcja, która wywołuje samą siebie w swoim ciele.

Kiedy funkcja rekurencyjna jest wywoływana, każde jej wywołanie tworzy nowy egzemplarz tej samej funkcji, który ma swoje własne zmienne i wykonuje swoje własne obliczenia. Wywołanie rekurencyjne kończy się w momencie, gdy zostanie spełniony warunek kończący, który zazwyczaj jest określony przez programistę, aby uniknąć nieskończonej rekurencji.

Rekurencja jest często stosowana w programowaniu, ponieważ pozwala na bardziej zwięzłe i eleganckie rozwiązanie wielu problemów, szczególnie tych, które mają strukturę podobną do drzewa lub listy. Przykłady algorytmów opartych na rekurencji to np. szybkie sortowanie (*quicksort*) i sortowanie przez scalanie (*merge sort*).

Warto jednak pamiętać, że rekurencja może prowadzić do bardzo głębokiej stosu wywołań funkcji, co może prowadzić do przekroczenia pamięci i problemów z wydajnością programu. Dlatego w niektórych przypadkach, zwykłe pętle mogą być lepszym rozwiązaniem niż rekurencja.

Oto przykładowy kod w języku Python, który implementuje algorytm silni rekurencyjnie:

Przykład w Python:

```
def silnia(n):
    if n == 1:
        return 1
    else:
        return n * silnia(n-1)
```

Powyższy kod oblicza silnię liczby `n` za pomocą rekurencji. Funkcja wywołuje samą siebie z argumentem mniejszym o 1, aż do osiągnięcia wartości 1. Gdy `n` jest równe 1, zwracana jest wartość 1.

Bazy danych

Przechodzimy do kolejnego bardzo dużego zagadnienia jakim są bazy danych!

Po co są bazy danych i jakie są ich podstawowe funkcje?

Bazy danych są narzędziami do przechowywania, organizowania i zarządzania dużymi ilościami danych. Służą one do przechowywania informacji, które są wykorzystywane przez różne aplikacje i systemy, takie jak systemy ERP, CRM, systemy finansowe czy aplikacje internetowe. Bazy danych są szeroko wykorzystywane w dziedzinach takich jak biznes, nauka, edukacja, medycyna i wiele innych.

Podstawową funkcją baz danych jest umożliwienie łatwego przechowywania, organizowania i zarządzania danymi. Bazy danych zapewniają strukturę dla danych, dzięki czemu użytkownicy mogą łatwo przeszukiwać, sortować i wyszukiwać informacje. Ponadto, bazy danych pozwalają na wykonywanie różnych operacji na danych, takich jak dodawanie, usuwanie, modyfikowanie, grupowanie i sumowanie.

Innymi funkcjami baz danych są:

1. **Zabezpieczanie danych:** bazy danych zapewniają różne poziomy zabezpieczeń danych, takie jak uwierzytelnianie użytkowników, autoryzacja dostępu do danych i szyfrowanie.
2. **Udostępnianie danych:** bazy danych umożliwiają łatwe udostępnianie danych między różnymi aplikacjami i systemami.
3. **Współbieżność:** bazy danych pozwalają na jednoczesny dostęp do danych przez wielu użytkowników, co jest szczególnie ważne w przypadku dużych systemów.
4. **Integrowanie danych:** bazy danych umożliwiają integrację danych z różnych źródeł, takich jak różne aplikacje czy systemy, co pozwala na łatwe tworzenie raportów i analiz.
5. **Tworzenie kopii zapasowych:** bazy danych umożliwiają tworzenie kopii zapasowych danych, co jest ważne w przypadku awarii systemu lub przypadkowej utraty danych.

Zatem wychodzi na to, że bazy danych są narzędziami do przechowywania, organizowania i zarządzania dużymi ilościami danych. Ich podstawową funkcją jest umożliwienie łatwego przechowywania, organizowania i zarządzania danymi, ale mają one także wiele innych funkcji, takich jak zabezpieczanie danych, udostępnianie danych, integracja danych i tworzenie kopii zapasowych.

Czy są rodzaje baz danych? Czy baza danych to po prostu baza danych?

Tak, istnieją różne rodzaje baz danych, a każdy z nich ma swoje charakterystyczne cechy i zastosowania w zależności od potrzeb i wymagań użytkownika.

Najbardziej podstawowe rodzaje baz danych to:

1. **Relacyjne bazy danych (RDBMS):** najbardziej powszechne i popularne typy baz danych, które organizują dane w tabelach związków między nimi. Przykładami relacyjnych baz danych są MySQL,

Oracle, SQL Server i PostgreSQL.

2. **Bazy danych NoSQL:** alternatywna koncepcja, która zakłada przechowywanie danych bez stosowania schematu tabeli. Bazy danych NoSQL umożliwiają przechowywanie dużych ilości danych niestrukturalnych, takich jak dokumenty, grafy lub klucze/wartości. Przykładami NoSQL są MongoDB, Cassandra i Redis.
3. **Bazy danych obiektowe:** przechowują dane w formie obiektów, które są powiązane z innymi obiektami w bazie danych. Bazy danych obiektowe są stosowane w językach programowania obiektowego, takich jak Java i C++.

Wszystkie te rodzaje baz danych różnią się między sobą strukturą przechowywania danych, sposobem organizacji danych, dostępnością danych, wydajnością i zastosowaniem.

Baza danych to ogólne pojęcie określające zorganizowany i udostępniany w sposób ciągły zbiór informacji, które można przetwarzać i analizować. Jednakże, rodzaje baz danych mogą znacznie się różnić między sobą, w zależności od ich struktury i cech.

Jak ma się baza danych do backendu?

Baza danych i backend są ze sobą ściśle powiązane w architekturze systemów informatycznych. Backend to część systemu, która odpowiada za przetwarzanie i obsługę żądań użytkowników oraz komunikację między interfejsem użytkownika (frontendem) a bazą danych.

W typowej architekturze aplikacji webowej, frontend odpowiada za prezentację danych dla użytkownika i interakcję z nim poprzez interfejs graficzny lub interfejs API. Backend z kolei przyjmuje żądania użytkownika, przetwarza je i odpowiada na nie, a także zarządza i przetwarza dane w bazie danych.

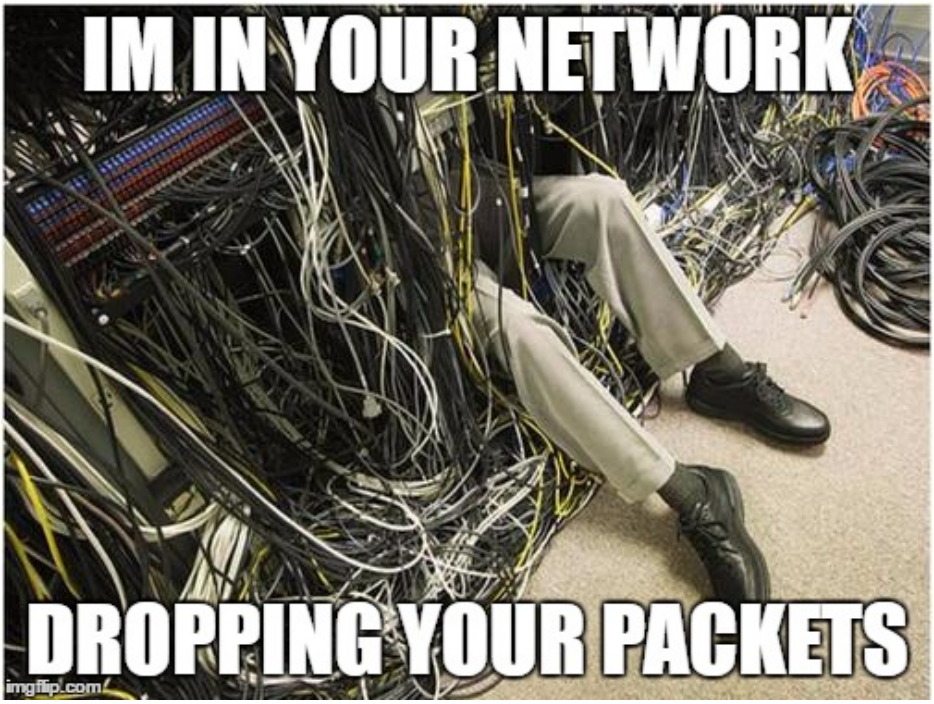
Baza danych jest elementem, w którym przechowywane są wszystkie dane, z których korzysta aplikacja. Backend odpowiada za pobieranie i zapisywanie danych w bazie danych, a także za ich przetwarzanie w celu wykonania żądań użytkownika. Backend może również przetwarzać dane na podstawie określonych zasad biznesowych, tj. zasad definiujących sposób działania aplikacji, co pozwala na uzyskanie odpowiedniego wyniku dla użytkownika.

Baza danych i backend są ze sobą powiązane w sposób, że backend odpowiada za przetwarzanie i obsługę danych przechowywanych w bazie danych, a baza danych zapewnia przechowywanie i zarządzanie tymi danymi. Bez bazy danych backend nie miałby dostępu do danych i nie mógłby obsługiwać żądań użytkownika, a bez backendu baza danych nie miałaby sposobu na przetworzenie danych i udostępnienie ich użytkownikom.

Sieci komputerowe

Jeden komputer — słaby. Wiele komputerów — silne. Tą słabą parafrazą cytatu z *"Planety Małp"* chciałbym zaprosić was do kolejnej części naszego programistycznego kompendium, w której skupimy się na łączeniu ze sobą komputerów w sieci. Te małe i duże, lokalne i globalne, a czym one się charakteryzują, dowiesz się, nasz drogi czytelniku, już za moment.

Jakie są podstawowe zasady działania sieci komputerowych?



Obraz 19. Źródło: https://imgflip.com/

No właśnie. Po co w ogóle łączyć ze sobą komputery w sieci? I czym te sieci tak właściwie są? Tak naprawdę sieci komputerowe to zbiór połączonych ze sobą urządzeń, takich jak komputery, drukarki, serwery i urządzenia mobilne, które pozwalają na wymianę danych i zasobów między nimi. Istnieją podstawowe zasady działania sieci komputerowych, które pozwalają na skuteczną i bezpieczną wymianę danych i zasobów między urządzeniami. Poniżej postaram się przybliżyć Ci kilka podstawowych zasad działania sieci komputerowych:

1. **Protokoły sieciowe:** sieci komputerowe wykorzystują protokoły sieciowe, czyli zestawy reguł, które określają sposób przesyłania danych między urządzeniami. Protokoły sieciowe określają sposób formatowania danych, kodowania i dekodowania, identyfikacji urządzeń i innych aspektów wymiany danych między urządzeniami.
2. **Adresy IP:** każde urządzenie w sieci komputerowej posiada unikalny adres IP, który pozwala na identyfikację i adresowanie urządzeń w sieci. Adresy IP są wykorzystywane w protokołach sieciowych, takich jak TCP/IP, do przesyłania danych między urządzeniami.
3. **Topologie sieciowe:** sieci komputerowe mogą być zorganizowane w różne topologie, takie jak topologia gwiazdy, siatki lub drzewa. Topologia sieciowa określa sposób, w jaki urządzenia są połączone ze sobą i jak dane są przesyłane między nimi.
4. **Bezpieczeństwo sieci:** bezpieczeństwo sieci jest bardzo ważne w sieciach komputerowych. Sieci komputerowe mogą być narażone na wiele rodzajów ataków, takich jak ataki hakerskie, wirusy i malware. W celu ochrony sieci komputerowej przed takimi atakami, należy zastosować różne techniki i narzędzia, takie jak hasła, zabezpieczenia sieciowe, oprogramowanie antywirusowe i wiele innych.
5. **Przepustowość sieci:** przepustowość sieci określa ilość danych, która może być przesłana między urządzeniami w sieci w określonym czasie. Przepustowość sieci zależy od wielu czynników, takich

jak topologia sieci, szybkość łącza internetowego i liczba urządzeń w sieci.

Więc sieci komputerowe opierają się na zasadach protokołów sieciowych, adresów IP, topologii sieciowej, bezpieczeństwa sieci i przepustowości sieci, które pozwalają na skuteczną i bezpieczną wymianę danych i zasobów między urządzeniami. Więc po co tak naprawdę łączymy komputery w sieci? Aby uzyskać dostęp do znacznie większej ilości zasobów. Skoro mówi się, że co dwie głowy to nie jedna, to pomyśl, co zamiast jednego komputera może zdziałać ich tuzin, setka lub milion.

Jakie są podstawy działania internetu?

Internecie, internecie — Ty jesteś jak zdrowie… Jak bardzo jesteś dla nas ważny, ten się tylko dowie, kto dostęp do Ciebie na toalecie straci… Internet to globalna sieć komputerowa, która pozwala na połączenie między sobą milionów urządzeń na całym świecie. Podstawą działania Internetu są trzy główne elementy:

1. **Sieć globalna:** Internet jest zbudowany na globalnej sieci komputerowej, która umożliwia połączenie milionów urządzeń na całym świecie. Sieć globalna składa się z wielu sieci regionalnych, krajowych i międzynarodowych, które łączą ze sobą urządzenia.
2. **Protokoły sieciowe:** Internet wykorzystuje protokoły sieciowe, czyli zestawy reguł, które określają sposób przesyłania danych między urządzeniami. Protokoły sieciowe określają sposób formatowania danych, kodowania i dekodowania, identyfikacji urządzeń i innych aspektów wymiany danych między urządzeniami.
3. **Adresacja IP:** każde urządzenie podłączone do Internetu posiada unikalny adres IP, który pozwala na jego identyfikację i adresowanie w sieci. Adresy IP są wykorzystywane w protokołach sieciowych, takich jak TCP/IP, do przesyłania danych między urządzeniami.

Działanie Internetu polega na przesyłaniu pakietów danych między urządzeniami za pomocą protokołów sieciowych. W celu przesyłania danych, każdy pakiet jest adresowany zgodnie z adresem IP docelowego urządzenia, a następnie przesyłany przez sieć globalną, aż dotrze do celu. W przypadku dużych plików lub danych, są one dzielone na wiele pakietów, które są przesyłane oddzielnie i następnie składane na końcu na docelowym urządzeniu.

W Internecie są również wykorzystywane różne usługi i protokoły, takie jak HTTP, FTP, SMTP, POP3, które pozwalają na przesyłanie danych, np. stron internetowych, plików lub poczty elektronicznej między urządzeniami.

Podstawą działania Internetu są globalna sieć komputerowa, protokoły sieciowe, adresacja IP i przesyłanie pakietów danych między urządzeniami. To pozwala na skuteczną wymianę danych, komunikację i korzystanie z różnych usług internetowych na całym świecie. Dzięki temu fantastycznemu wynalazkowi mamy natychmiastowy dostęp do naszych ulubionych filmów, muzyki, gier i memów z kotkami.

Czym różni się internet od sieci komputerowej?

A no właśnie! Internet i sieć komputerowa to dwa różne pojęcia, chociaż są ze sobą powiązane.

Sieć komputerowa to połączenie dwóch lub więcej urządzeń komunikacyjnych, takich jak komputery, serwery, drukarki, itp., które są połączone ze sobą w celu wymiany danych i zasobów. Sieć może być

zlokalizowana w jednym pomieszczeniu, budynku, kampusie lub obejmować wiele oddalonych geograficznie lokalizacji. Sieć umożliwia komunikację między węzłami i udostępnianie zasobów między nimi, takich jak pliki, drukarki, urządzenia pamięci masowej itp.

Internet to globalna sieć połączeń między różnymi sieciami komputerowymi, która umożliwia komunikację i wymianę informacji między milionami urządzeń na całym świecie. Internet jest oparty na protokole TCP/IP (Transmission Control Protocol/Internet Protocol), który umożliwia wymianę danych między różnymi sieciami i urządzeniami.

Sieć komputerowa to lokalna sieć połączeń między urządzeniami komunikacyjnymi w jednym pomieszczeniu, budynku lub kampusie, podczas gdy internet to globalna sieć połączeń między różnymi sieciami komputerowymi na całym świecie. Sieć komputerowa jest częścią internetu i może być połączona z internetem, aby umożliwić dostęp do zasobów i usług dostępnych w internecie. W uproszczeniu: sieć komputerowa — kilka (kilkanaście / kilkadziesiąt) komputerów niedaleko siebie; internet — miliony komputerów na całym świecie.

Czym są protokoły sieciowe i do czego są potrzebne?

Hatetepe, Eftepe Sretetete… Nie bój nic, zaraz wszystko stanie się jasne. Protokoły sieciowe to zestawy reguł, zasad i procedur, które określają, jak urządzenia komunikacyjne, takie jak komputery, serwery, routery, modemy i inne, wymieniają między sobą dane w sieci komputerowej. Protokoły sieciowe są niezbędne do zapewnienia zgodności i koordynacji między urządzeniami w sieci oraz umożliwiają standardowe procedury wymiany danych. Protokoły sieciowe są potrzebne do wielu różnych zastosowań, w tym:

1. **Przesyłanie danych**: protokoły sieciowe zapewniają standardowe procedury przesyłania danych między urządzeniami, takie jak przesyłanie plików, wiadomości e-mail, strony internetowe itp.
2. **Kontrola błędów:** protokoły sieciowe określają procedury wykrywania i korygowania błędów podczas przesyłania danych.
3. **Zarządzanie siecią:** protokoły sieciowe umożliwiają monitorowanie i zarządzanie siecią, w tym konfigurowanie urządzeń sieciowych, ustalanie priorytetów ruchu sieciowego, diagnozowanie problemów sieciowych itp.
4. **Ustanawianie połączenia:** protokoły sieciowe umożliwiają urządzeniom w sieci komunikację między sobą, nawiązywanie i zrywanie połączeń.
5. **Zabezpieczanie sieci:** protokoły sieciowe są wykorzystywane do zabezpieczania sieci przed nieautoryzowanym dostępem, atakami sieciowymi i innymi zagrożeniami.

Najczęściej stosowane protokoły sieciowe to TCP/IP (Transmission Control Protocol/Internet Protocol), HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), POP (Post Office Protocol) i IMAP (Internet Message Access Protocol).

W skrócie, protokoły sieciowe to standardowe procedury i zasady, które zapewniają koordynację między urządzeniami w sieci, umożliwiają przesyłanie danych, kontrolę błędów, zarządzanie siecią, ustanawianie połączenia i zabezpieczanie sieci. Bez protokołów sieciowych, urządzenia w sieci nie mogłyby się ze sobą porozumieć i wymieniać informacji. Państwa na świecie mają swoje prawa i procedury, aby ludzie w nich mogli liczyć na bezpieczeństwo i porządek.

Do tego samego służą protokoły sieciowe. Dzięki nim masz pewność, że wrzucany przez Ciebie na

socjalki memesek z pieskiem zostanie poprawnie zapisany na serwerze i wyświetlony u wszystkich Twoich znajomych oraz, że piosenka, którą ściągasz na komputer nie przywlecze ze sobą wirusa, który potem połączy się z Twoją mikrofalówką i przejdzie Ci do kotleta.

Czym jest serwer i do czego jest potrzebny?

Serwer to komputer, który znajduje się setki kilometrów od Ciebie i z którego pobierane są dane na Twój komputer, gdy korzystasz z internetu. Serwer to program lub urządzenie komputerowe, które udostępnia usługi lub zasoby dla innych urządzeń, zwanych klientami. Serwer jest zazwyczaj zainstalowany na komputerze lub urządzeniu sieciowym, które jest zaprojektowane do przetwarzania dużych ilości danych i zapewnienia stabilnego i nieprzerwanego działania usług. Serwer jest potrzebny do wielu różnych zastosowań, w tym:

1. **Udostępnianie zasobów:** serwer może udostępniać zasoby takie jak pliki, drukarki, urządzenia pamięci masowej, bazy danych itp. dla klientów w sieci.
2. **Przetwarzanie danych:** serwer może przetwarzać i analizować duże ilości danych w celu uzyskania wyników lub generowania raportów.
3. **Udostępnianie usług:** serwer może udostępniać różnego rodzaju usługi, takie jak usługi internetowe, pocztowe, FTP, VoIP itp.
4. **Hostowanie stron internetowych:** serwer może hostować strony internetowe, umożliwiając ich dostępność dla użytkowników w internecie.
5. **Zarządzanie siecią:** serwer może zarządzać i kontrolować sieć, monitorując jej wydajność, konfigurując urządzenia, zabezpieczając sieć itp.

W przypadku aplikacji internetowych, serwer jest odpowiedzialny za udostępnianie treści i usług dla klientów, takich jak przetwarzanie zapytań HTTP i dostarczanie treści w formacie HTML, CSS i JavaScript. Serwer jest często wykorzystywany w połączeniu z bazami danych, aby umożliwić przechowywanie danych i przetwarzanie żądań klienckich.

Czyli serwer jest kluczowym elementem infrastruktury sieciowej, który umożliwia udostępnianie i przetwarzanie danych, udostępnianie usług i zasobów, a także zarządzanie siecią. Wszystko, co wyświetla się w Twojej przeglądarce podczas korzystania z internetu jest na bieżąco ściągane z serwera na Twój komputer.

Podstawy projektów programistycznych

O, matko! Chyba jakieś trudne rzeczy! Niekoniecznie. Jeżeli myślisz nad zostaniem programistą, bardzo ważne jest planowanie swoich projektów niezależnie, czy zamierzasz robić je samodzielnie lub w zespole. To pomoże Ci działać sprawnie, efektywnie i osiągać zamierzone cele. Ale zacznijmy od początku...

Jakie są podstawowe zasady projektowania projektów programistycznych?

Najpierw pomyśl, potem działaj! Złota zasada nie tylko w programowaniu, ale i w codziennym życiu. Oto kilka podstawowych zasad projektowania projektów programistycznych, które pomogą Ci usprawnić

Twoją pracę:

1. **Planuj i zaprojektuj przed implementacją:** Przed rozpoczęciem pisania kodu, warto poświęcić czas na stworzenie planu projektu i zaprojektowanie architektury systemu. Planowanie i projektowanie pozwala uniknąć błędów i skrócić czas potrzebny na testowanie i poprawianie błędów.
2. **Użyj odpowiednich narzędzi:** W zależności od projektu i języka programowania, użyj odpowiednich narzędzi, aby zoptymalizować swój workflow. Na przykład, jeśli piszesz kod w języku Python, warto skorzystać z narzędzi takich jak PyCharm lub Jupyter Notebook.
3. **Działaj modułowo:** Dzielenie kodu na mniejsze moduły ułatwia zarządzanie i testowanie projektu. Warto również unikać powtarzalnego kodu i korzystać z zewnętrznych bibliotek, jeśli to możliwe.
4. **Utrzymuj czysty kod:** Stosowanie dobrych praktyk programistycznych i utrzymywanie czystego kodu zwiększa czytelność i ułatwia późniejszą modyfikację projektu.
5. **Dokumentuj projekt:** Dokumentacja projektu jest ważna, aby inni członkowie zespołu lub przyszli programiści mogli łatwo zrozumieć, co zostało zrobione i jak działa projekt. Warto również pisać komentarze do kodu, aby ułatwić zrozumienie kodu innym programistom.
6. **Testuj i poprawiaj błędy:** Testowanie projektu na bieżąco pozwala na szybkie wykrywanie i usuwanie błędów. Warto również korzystać z narzędzi do automatycznego testowania kodu, takich jak np. pytest.
7. **Pracuj w zespole:** Jeśli pracujesz w zespole, warto ustalić zasady dotyczące pracy z kodem i korzystania z narzędzi, takich jak repozytorium kodu (np. GitHub). Praca w zespole ułatwia również dzielenie się wiedzą i rozwiązywanie problemów.

Jak widzisz, nie ma tu żadnych czarów. Wystarczy stosować się do kilku prostych zasad, które odwołują się do zdrowego rozsądku. Dzięki temu nasza praca będzie bardziej wydajna i uporządkowana.

W jaki sposób organizuje się projekty programistyczne?

Organizacja projektów programistycznych może się różnić w zależności od specyfiki projektu, skali i rodzaju zespołu programistycznego. Oto jednak kilka powszechnych praktyk, które pomagają w organizacji projektów programistycznych:

1. **Określenie celów projektu:** Pierwszym krokiem do organizacji projektu programistycznego jest określenie celów, które ma on osiągnąć. Cel ten powinien być wyraźny, jasny i możliwy do zmierzenia.
2. **Utworzenie zespołu programistycznego:** W zależności od rozmiaru projektu, można utworzyć zespół programistyczny składający się z jednej lub wielu osób. Ważne jest, aby każda osoba w zespole miała określone role i zadania, a także jasne oczekiwania co do ich pracy.
3. **Określenie etapów projektu:** Projekt programistyczny może być podzielony na kilka etapów, takich jak planowanie, projektowanie, implementacja, testowanie i wdrażanie. Dzięki podziałowi projektu na etapy łatwiej jest określić postępy w pracy i monitorować stan projektu.
4. **Używanie narzędzi do zarządzania projektem:** Istnieje wiele narzędzi do zarządzania projektami programistycznymi, takich jak Trello, Asana, czy Jira. Narzędzia te pomagają w zarządzaniu zadaniami, monitorowaniu postępów projektu i organizowaniu pracy zespołu.
5. **Utrzymywanie kontaktu z klientem:** W przypadku projektów programistycznych dla klientów,

ważne jest, aby utrzymywać stały kontakt z klientem i informować go o postępach w pracy. Dzięki temu można łatwiej dopasować projekt do oczekiwań klienta.

6. **Dbałość o dokumentację projektu:** Dokumentacja projektu jest ważna dla zrozumienia, jak działa projekt oraz jak go rozwijać w przyszłości. Należy pamiętać o regularnym aktualizowaniu dokumentacji i umieszczeniu jej w repozytorium kodu projektu.
7. **Regularne spotkania zespołu:** Spotkania zespołu są ważne dla monitorowania postępów projektu, omawiania problemów oraz dzielenia się pomysłami. Spotkania te mogą odbywać się codziennie lub raz w tygodniu, w zależności od potrzeb zespołu.

Pamiętaj, że dobre zarządzanie projektami programistycznymi jest kluczowe dla zapewnienia skutecznego i efektywnego procesu tworzenia oprogramowania. Jeśli każdy w zespole rzuci się w wir pracy bez wyraźnej i zaplanowanej komunikacji między sobą, na dłuższą metę nie przynosi to nigdy niczego dobrego.

Czym jest Agile?

A, gile! Te takie zielone z nosa? Też, ale tutaj skupimy się na trochę czym innym. Agile to podejście do zarządzania projektami, w szczególności projektami oprogramowania, które kładzie nacisk na elastyczność, szybkość i efektywność. Metodologia Agile opiera się na iteracyjnym podejściu do projektowania, w którym projekt jest dzielony na mniejsze kawałki (iteracje) i każda iteracja dostarcza wartości dla klienta. Agile zakłada, że projekty oprogramowania są zawsze niepewne i wymagają ciągłego dopasowywania i dostosowywania do zmieniających się wymagań klienta i środowiska biznesowego. W metodologii Agile wyróżnia się kilka wartości i zasad, które stanowią podstawę podejścia Agile, a są to:

1. Indywidualne kontakty i interakcje między ludźmi ponad procesami i narzędziami.
2. Działające oprogramowanie ponad wyczerpującą dokumentacją.
3. Współpraca z klientem ponad negocjowanie umów.
4. Odpowiedź na zmiany ponad zastosowanie planów.

Metodologia Agile ma wiele różnych praktyk, w tym Scrum, Kanban, Lean i XP (Extreme Programming), które mogą być stosowane w zależności od potrzeb projektu i preferencji zespołu. Agile jest popularnym podejściem do zarządzania projektami oprogramowania, ponieważ pozwala na elastyczne dostosowywanie projektu do zmieniających się wymagań klienta, a jednocześnie zapewnia stałe dostarczanie wartości.

Czym jest Scrum?

Jak nie robisz tego sam, zawsze przyda Ci się Skram! Scrum to podejście zwinne do zarządzania projektami, które opiera się na iteracyjnym i inkrementalnym podejściu do projektowania. Scrum zakłada, że zespół programistów powinien pracować razem jako zespół samodzielny i samodzielnie podejmować decyzje dotyczące projektu. Scrum składa się z kilku elementów:

1. **Product backlog:** lista wymagań produktu, która jest stale aktualizowana i priorytetowa. Wymagania te są opisywane przez właściciela produktu (product owner), a zespół programistów decyduje, które wymagania zostaną wzięte pod uwagę w kolejnej iteracji.

2. **Sprint:** krótki okres, zwykle trwający od 1 do 4 tygodni, w którym zespół programistów pracuje nad wybranymi wymaganiami z product backlogu.
3. **Sprint backlog:** lista zadań, które zespół programistów musi wykonać w trakcie sprintu.
4. **Daily Scrum:** krótkie, codzienne spotkania zespołu programistów, podczas których omawiane są postępy w pracy, problemy i cele na kolejny dzień.
5. **Sprint review:** spotkanie, podczas którego zespół programistów prezentuje swoje postępy klientowi i omawia, co zostało zrobione w trakcie sprintu.
6. **Sprint retrospective:** spotkanie, podczas którego zespół programistów omawia swoje doświadczenia i decyduje, co można zrobić lepiej w kolejnym sprincie.

Scrum jest popularnym podejściem do zarządzania projektami programistycznymi, ponieważ pozwala na elastyczne dostosowywanie projektu do zmieniających się wymagań klienta, a jednocześnie zapewnia stałe dostarczanie wartości.

Czym są systemy kontroli wersji i do czego są wykorzystywane?

Kto to tak spartolił?! Znasz to — pracujecie w zespole, ktoś popsuł coś, co do tej pory działało, podejrzewasz Stefana ale nie masz dowodów, nikt się do niczego nie przyznaje. Dzięki Systemom Kontroli Wersji możesz nie tylko udowodnić winę Stefana ale co ważniejsze, przywrócić stan aplikacji sprzed jego ingerencji. Bo Systemy kontroli wersji (ang. Version Control System, VCS) to narzędzia, które pozwalają na śledzenie i kontrolowanie zmian w kodzie źródłowym lub innych plikach. Dzięki systemowi kontroli wersji możliwe jest śledzenie historii zmian, cofanie do poprzednich wersji plików, porównywanie zmian między różnymi wersjami plików, a także umożliwienie wielu programistom pracy nad tym samym kodem równocześnie. Systemy kontroli wersji są niezbędne w projektach programistycznych, ponieważ pozwalają na:

1. **Utrzymanie historii zmian:** Systemy kontroli wersji pozwalają na śledzenie i przechowywanie historii zmian w kodzie źródłowym lub innych plikach. Dzięki temu można cofnąć się do poprzednich wersji plików, jeśli zajdzie taka potrzeba.
2. **Współpracę w zespole:** Systemy kontroli wersji umożliwiają wielu programistom pracę nad tym samym kodem równocześnie, a także kontrolują, kto wprowadził określone zmiany i kiedy.
3. **Bezpieczeństwo i ochronę przed utratą danych:** Systemy kontroli wersji umożliwiają tworzenie kopii zapasowych i zabezpieczanie kodu źródłowego przed utratą danych w przypadku awarii sprzętu lub oprogramowania.
4. **Zarządzanie wydaniami:** Systemy kontroli wersji pozwalają na kontrolowanie, które zmiany zostaną uwzględnione w kolejnej wersji produktu lub kodu.

Najpopularniejszymi systemami kontroli wersji są Git, Subversion (SVN) i Mercurial. Współczesne projekty programistyczne niemal zawsze korzystają z systemów kontroli wersji jako narzędzia do zarządzania kodem źródłowym i plikami projektowymi. Stefan już nigdy więcej się nie wywinie!

Testowanie napisanych programów

Obraz 20. Źródło: Testing in Production - Everything You Need To Know

U mnie działa... Po co to testować? Mam nadzieję, że nie wychodzisz z takiego założenia, bo testy są bardzo ważną częścią powstawania programów i zaraz postaram Ci się wyjaśnić dlaczego.

Czym są testy w programowaniu?

Testy w programowaniu to proces sprawdzania, czy program działa zgodnie z oczekiwaniami. Testy służą do weryfikowania poprawności działania kodu źródłowego oraz wykrywania błędów i uchybień w projekcie. W programowaniu wyróżnia się kilka rodzajów testów, które mogą być wykonywane na różnych etapach procesu tworzenia oprogramowania. Oto niektóre z najważniejszych rodzajów testów:

1. **Testy jednostkowe:** to testy, które sprawdzają pojedyncze jednostki kodu źródłowego, takie jak metody czy funkcje. Testy jednostkowe są zwykle automatyzowane i wykonują się bardzo szybko.
2. **Testy integracyjne:** to testy, które sprawdzają, jak poszczególne jednostki kodu źródłowego współpracują ze sobą i czy działają zgodnie z oczekiwaniami. Testy integracyjne są zwykle wykonywane automatycznie.
3. **Testy funkcjonalne:** to testy, które sprawdzają, czy program działa zgodnie z oczekiwaniami użytkownika. Testy funkcjonalne są zwykle wykonywane ręcznie, przez testera lub użytkownika końcowego.
4. **Testy wydajnościowe:** to testy, które sprawdzają, jak program działa pod obciążeniem i czy działa w odpowiednim czasie. Testy wydajnościowe są zwykle wykonywane automatycznie.
5. **Testy bezpieczeństwa:** to testy, które sprawdzają, czy program jest odporny na ataki i czy użytkownik nie może uzyskać dostępu do danych lub funkcjonalności, do których nie powinien mieć dostępu.

Testowanie jest ważnym etapem w procesie tworzenia oprogramowania, ponieważ pozwala na wykrywanie i usuwanie błędów oraz zapewnia, że program działa zgodnie z oczekiwaniami użytkowników. Automatyzacja testów pozwala na szybsze i bardziej efektywne testowanie oraz zmniejsza ryzyko popełnienia błędów przez testerów. Nie możesz pominąć procesu testowania, jeśli nie

chcesz, aby Twoi użytkownicy powyrywali sobie włosy z głowy podczas używania Twojej aplikacji.

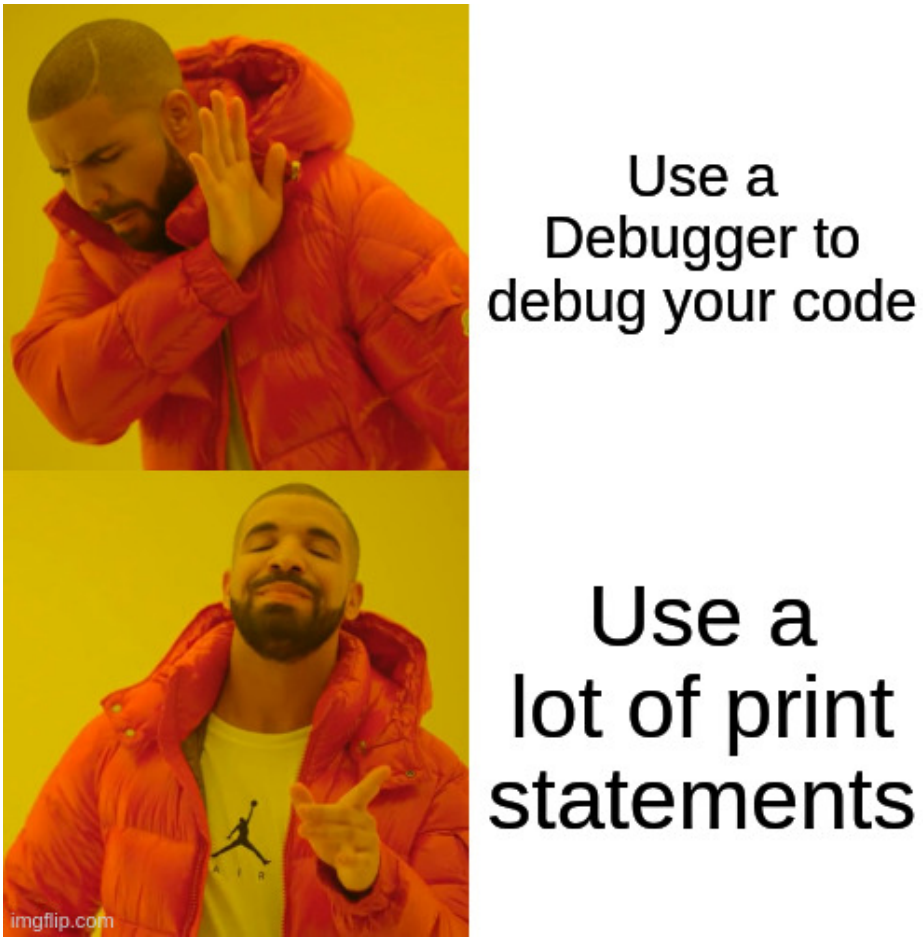
Po co testuje się programy?

A komu to potrzebne? Na pewno włosom użytkowników Twoich programów. Testowanie jest bardzo ważnym etapem w procesie tworzenia oprogramowania. Poniżej postaram się przedstawić najważniejsze powody, dla których testuje się programy:

1. **Wykrywanie błędów i uchybień:** testowanie programów pozwala na wykrycie błędów i uchybień w kodzie źródłowym oraz na zidentyfikowanie problemów, które mogą spowodować nieprawidłowe działanie programu.
2. **Poprawa jakości oprogramowania:** testowanie programów jest kluczowym elementem poprawy jakości oprogramowania. Dzięki testom można wykryć i usunąć błędy, co przekłada się na poprawę funkcjonalności, wydajności i bezpieczeństwa programu.
3. **Optymalizacja wydajności:** testowanie programów pozwala na zidentyfikowanie problemów związanych z wydajnością i optymalizację działania programu.
4. **Zwiększenie niezawodności:** testowanie programów pozwala na zwiększenie niezawodności programu i zmniejszenie ryzyka wystąpienia błędów.
5. **Umożliwienie dostarczania wartości:** testowanie programów pozwala na dostarczenie wartości użytkownikowi, ponieważ program działa zgodnie z oczekiwaniami i spełnia jego wymagania.
6. **Utrzymanie reputacji:** testowanie programów jest ważne dla reputacji firmy, ponieważ umożliwia dostarczenie wysokiej jakości oprogramowania, które działa zgodnie z oczekiwaniami klienta.
7. **Optymalizacja kosztów:** testowanie programów pozwala na zmniejszenie kosztów związanych z naprawą błędów i poprawą jakości programu.

Testowanie pozwala na wykrycie błędów i uchybień, poprawę jakości, optymalizację wydajności i zwiększenie niezawodności programu. Pamiętaj — zadowolony użytkownik, to taki, który po skorzystaniu z Twojej apki ma tyle samo włosów na głowie, co przed skorzystaniem z niej.

Czym jest debugowanie?



Obraz 21. Źródło: https://twitter.com/PR0GRAMMERHUM0R

Program jest jak mieszkanie. Zaniedbanie go powoduje mnożenie się w nim różnego rodzaju robaczków. Debugowanie to proces znajdowania i eliminowania błędów lub uchybień w kodzie źródłowym programu. Proces ten ma na celu sprawdzenie, co jest nie tak w działaniu programu, znalezienie przyczyny błędu i naprawienie go.

Podczas debugowania, programista może korzystać z różnych narzędzi do analizowania kodu źródłowego, np. debuggerów, logów, wyjątków, testów jednostkowych, itp. Debugowanie może być wykonywane na różnych etapach procesu tworzenia oprogramowania, np. podczas tworzenia nowych funkcjonalności, testów integracyjnych, wydajnościowych, itp.

Proces debugowania może być skomplikowany i wymagać dużo czasu, szczególnie w przypadku trudnych do wykrycia błędów. W związku z tym, istnieją różne techniki i strategie debugowania, które pomagają programistom zoptymalizować proces debugowania i znaleźć błędy szybciej i efektywniej.

Współcześnie wiele środowisk programistycznych oferuje rozbudowane narzędzia do debugowania, takie jak debugger, który umożliwia stopowanie programu w określonych miejscach, wykonywanie kodu linia po linii, przeglądanie wartości zmiennych, a także badanie stosu wywołań. To ułatwia proces debugowania, szczególnie dla bardziej skomplikowanych programów.

Zakończenie

Doszliśmy do końca tej książki. Ta podróż była pełna wyzwań i trudności, ale teraz, gdy spojrzysz na początek tej książki, możesz zobaczyć jak dużo udało Ci się już dowiedzieć.

Programowanie to nie tylko pisanie kodu. To również współpraca, kreatywność i wyobraźnia. W świecie programowania nie ma jednej właściwej drogi, ale wiele możliwości i podejść. Każdy z nas może znaleźć swoje własne pasje i zastosowania dla tych narzędzi.

Teraz, gdy zakończyliśmy to wprowadzenie, możesz śmiało zmierzać w kierunku naszego kursu. To dopiero początek naszej drogi w świecie kodu. Niech Twój umysł pozostanie otwarty na nowe wyzwania i niech Twoja ciekawość prowadzi Cię do nowych odkryć.

Najważniejsze, co musisz pamiętać, to nie bać się programowania. Pisanie kodu może być frustrujące i trudne, ale to część procesu nauki. Najważniejsze to nie poddawać się, być cierpliwym i wierzyć w swoje umiejętności. Przygotuj się, że w trakcie tego kursu będziesz się dużo frustrować.

Wreszcie, chcielibyśmy podziękować wszystkim czytelnikom, którzy podjęli wyzwanie nauki programowania razem z nami. Mamy nadzieję, że uda nam się spłacić zaciągnięty kredyt zaufania.

Zanim pójdziesz dalej, chcemy przekazać Ci kilka ważnych rad. Po pierwsze, pamiętaj, że nauka programowania to proces ciągły. Technologia rozwija się w zawrotnym tempie, nowe języki i narzędzia pojawiają się każdego dnia. Dlatego ważne jest, aby być na bieżąco i kontynuować rozwijanie swoich umiejętności.

Nie bój się eksperymentować i popełniać błędy. Często to właśnie przez próbowanie i niepowodzenia uczysz się najwięcej. Każdy błąd to szansa na naukę i doskonalenie naszych umiejętności. Niech Twoje porażki staną się motorem napędzającym Cię do dalszego rozwoju.

Pamiętaj również, że programowanie to nie tylko samotna praca. Otaczaj się ludźmi o podobnych zainteresowaniach, dołączaj do społeczności programistycznych, uczestnicz w hackathonach i konferencjach. Wymiana doświadczeń i współpraca z innymi programistami może przynieść nie tylko nowe perspektywy, ale także inspirację i przyjaźnie na długie lata.

Nie zapominaj również o równowadze. Programowanie może być wciągające i czasochłonne, ale ważne jest, aby dbać o swoje zdrowie fizyczne i emocjonalne. Znajdź czas na aktywność fizyczną, odpoczynek i spędzanie czasu z bliskimi. To wzmocni Twoją produktywność i pozwoli czerpać radość z tego, co robisz.

Na koniec, bądź dumny/-a z tego, czego się nauczysz i co osiągniesz. Programowanie to umiejętność, która otwiera drzwi do nieskończonych możliwości. Niezależnie od tego, czy tworzysz aplikacje, rozwijasz gry czy analizujesz dane, Twoja praca ma znaczenie. Wykorzystuj swoje umiejętności w twórczy i odpowiedzialny sposób, przyczyniając się do rozwoju społeczeństwa i poprawy jakości życia.

Niech koniec tej książki będzie początkiem Twojej własnej historii w świecie programowania. Życzymy Ci powodzenia na ścieżce zajavka.

Team Zajavka.pl