



Wprowadzenie do programowania obiektowego



AGENDA:

1. PRZEDSTAWIENIE
2. PARADYGMATY PROGRAMOWANIA
3. JĘZYKI PROGRAMOWANIA
4. OBIEKTY I KLASY
5. TWORZENIE OBIEKTÓW W PROGRAMIE



POZNAN UNIVERSITY OF TECHNOLOGY

PROGRAMOWANIE OBIEKTOWE
WSTĘP DO PROGRAMOWANIA OBIEKTOWEGO
Serhii Baraban

PRZEDSTAWIENIE



Moje doświadczenie

Intelligent module of browser management system of IT-projects Trello

In the article was proposed a programmatic solution in the form of an intelligent module for expanding the functionality of a browser-based management system for IT-projects - Trello, based on the use of its API. To select data from the boards, lists, cards, functions were developed using procedural programming methods. During the development of the intellectual module, a UML-class diagram of the intellectual module developed according to the existing py-trello library. For aggregation of data, selected from the boards, lists, cards, the modern Pandas library is used. The Pandas features such as reading, writing to a Microsoft Excel file, data grouping, reloading the table index, arithmetic adding and multiplication were used. Approval of the results confirmed the feasibility of the development of the implementation of monthly reporting and calculation of employee productivity metrics.

Development of a progressive web application with a convolutional neural network for image recognition

In this paper the technologies for creating web applications are analyzed, in result Progressive Web App as the most suitable for solving the tasks is selected. The peculiarities of the use of intelligent technologies for the problem of image recognition are investigated. Emphasis is placed on methods that use the TensorFlow neural network library. The own model of convolutional neural network for image recognition has been created. The dataset «The Quick, Draw! Dataset» from Google is selected for model training. It has been determined that a progressive web application provides the ability to provide the resulting sample to the user faster than analogues. The result of comparing the speed of the developed and analog applications is illustrated.

Moje
doświadczenie w
polsce :

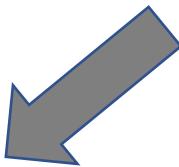




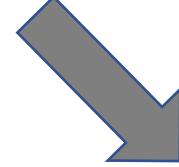
Aktualny stan :



Flask



djang^o



+

+



SQLAlchemy



PostgreSQL



Opis przedmiotu :

1. Nauczenie studentów zasad tworzenia uniwersalnych modułów programowych zdatnych do wielokrotnego wykorzystania w różnych projektach programistycznych i łatwych w rozwoju i pielęgnacji, poprzez zastosowanie unikalnych rozwiązań dostępnych w językach obiektowych, które sprzyjają tworzeniu programów komputerowych o takich cechach. Ponadto celem jest nauczenie studentów tworzenia własnych bogatych semantycznie i uniwersalnych abstrakcyjnych typów danych.

2. Rozwijanie u studentów umiejętności projektowania i tworzenia systemów informatycznych o poprawnej architekturze, to jest takiej, która charakteryzuje się spoistością składowych modułów programowych i luźnych związków między tymi modułami.

3. Kształtowanie u studentów umiejętności komunikacji podczas niezależnego tworzenia modułów programów komputerowych, które mają być skomponowane w jedną całość. Ponadto uzyskanie umiejętności wyszukiwania optymalnych, gotowych i dostępnych komponentów, do wykorzystania we własnych złożonych programach komputerowych.



Portret Współczesnego Programisty:

Współczesny programista to nie tylko osoba, która **zna języki programowania i potrafi pisać kod.**

To profesjonalista posiadający szereg umiejętności technicznych, organizacyjnych i interpersonalnych, które umożliwiają mu skuteczne funkcjonowanie w dynamicznym i często złożonym środowisku pracy.

Kluczowe umiejętności

umiejętności techniczne (**hard skills**)

umiejętności miękkie (**soft skills**)



Umiejętności Techniczne (Hard Skills):

1. Znajomość Języków Programowania:

- Współczesny programista powinien znać co najmniej kilka języków programowania (np. Python, Java, JavaScript) oraz rozumieć ich różnice i zastosowania.
- Znajomość kilku różnych paradymatów programowania, takich jak programowanie obiektowe, funkcyjne, czy proceduralne, zwiększa jego elastyczność i zdolność adaptacji do różnych projektów.

2. Algorytmy i Struktury Danych:

- Umiejętność projektowania wydajnych algorytmów oraz znajomość struktur danych (takich jak listy, drzewa, stosy, kolejki) jest kluczowa dla optymalizacji kodu i rozwiązywania złożonych problemów.
- Rozumienie złożoności obliczeniowej i znajomość technik optymalizacji kodu.

3. Znajomość Systemów Operacyjnych:

- Programista powinien mieć podstawową wiedzę o systemach operacyjnych, takich jak Windows, Linux czy macOS, oraz umiejętność pracy z terminaliem.
- Rozumienie podstaw zarządzania zasobami, procesami i pamięcią w systemie operacyjnym.



Umiejętności Techniczne (Hard Skills):

4. Praca z Systemami Kontroli Wersji (np. Git):

Zarządzanie kodem i wersjami projektu jest niezbędne w pracy zespołowej. Znajomość Gita i platform takich jak GitHub czy GitLab pozwala na współpracę z innymi programistami i sprawne zarządzanie kodem źródłowym.

5. Podstawy Pracy z Bazami Danych:

Umiejętność tworzenia, modyfikowania i zarządzania bazami danych SQL (np. PostgreSQL, MySQL) i NoSQL.

6. Zasady Bezpieczeństwa w Programowaniu.

7. Testowanie i Debugowanie:

Współczesny programista zna podstawowe techniki testowania, takie jak testy jednostkowe, integracyjne oraz automatyzacja testów.

Debugowanie i analiza błędów w kodzie to kluczowa umiejętność pozwalająca na szybkie rozwiązywanie problemów.

8. Zarządzanie Projektami i Narzędzia DevOps:

Znajomość narzędzi DevOps, takich jak Docker, Kubernetes, CI/CD (np. Jenkins), które wspierają automatyzację wdrażania aplikacji.



Umiejętności Miękkie (Soft Skills):

1. Umiejętność Pracy Zespołowej:

Współczesny programista często pracuje w zespołach, zarówno lokalnych, jak i zdalnych, co wymaga umiejętności współpracy, dzielenia się wiedzą i odpowiedzialności.

Umiejętność efektywnego komunikowania się z członkami zespołu, a także otwartość na konstruktywną krytykę.

2. Rozwiązywanie Problemów i Kreatywność:

Programista musi być zdolny do analitycznego myślenia i rozwiązywania problemów, często wymagających niestandardowego podejścia.

Kreatywność w podejściu do kodowania oraz znajdowanie optymalnych rozwiązań technicznych.

3. Adaptacja i Elastyczność:

Branża technologiczna szybko się zmienia, dlatego ważna jest otwartość na nowe technologie i chęć nauki. Zdolność do szybkiej adaptacji do nowych narzędzi, bibliotek czy frameworków oraz elastyczność w podejściu do zmieniających się wymagań projektu.

4. Umiejętności Komunikacyjne:

Umiejętność wyjaśnienia złożonych zagadnień technicznych w sposób zrozumiały dla osób nietechnicznych (np. menedżerów projektów, klientów).

Znajomość podstawowej terminologii, a także zdolność do efektywnego raportowania postępów i problemów.



Umiejętności Miękkie (Soft Skills):

5. Zarządzanie Czasem i Organizacja Pracy:

Zdolność do pracy w warunkach presji czasu oraz skutecznego zarządzania wieloma zadaniami. Planowanie pracy w taki sposób, aby realizować cele projektu i spełniać wyznaczone terminy.

6. Dbałość o Detale i Cierpliwość:

Wysoka dbałość o szczegóły pozwala uniknąć błędów i zwiększa jakość kodu. Cierpliwość w analizowaniu kodu i poszukiwaniu rozwiązań dla trudnych problemów technicznych.

7. Chęć Ciągłego Rozwoju i Nauki:

Branża IT wymaga stałego rozwoju, dlatego współczesny programista powinien regularnie poszerzać swoje umiejętności, uczestniczyć w kursach, czytać dokumentację i śledzić nowinki technologiczne. Budowanie własnego portfolio, tworzenie projektów open-source lub udział w konferencjach i spotkaniach branżowych to ważne elementy rozwoju kariery.



POZNAN UNIVERSITY OF TECHNOLOGY

PROGRAMOWANIE OBIEKTOWE
WSTĘP DO PROGRAMOWANIA OBIEKTOWEGO
Serhii Baraban

A co z językiem angielskim?



Rzeczywiście, znajomość języka angielskiego jest obecnie kluczową umiejętnością dla współczesnych programistów.

W dzisiejszym świecie technologii, większość dokumentacji technicznej, narzędzi, bibliotek oraz zasobów edukacyjnych dostępnych jest w języku angielskim.



Oto dlaczego znajomość angielskiego jest tak ważna dla współczesnego programisty:

1. Dostęp do Dokumentacji i Zasobów:

Większość oficjalnej dokumentacji i materiałów szkoleniowych (np. dla języków programowania takich jak Python, JavaScript, czy frameworków jak React) jest napisana po angielsku. Programiści, którzy znają ten język, mają łatwiejszy dostęp do informacji i mogą lepiej rozumieć niuanse techniczne.

2. Komunikacja w Zespołach Międzynarodowych:

Coraz więcej firm prowadzi działalność globalnie i zatrudnia specjalistów z różnych krajów. Angielski jest językiem uniwersalnym, który umożliwia sprawną komunikację w międzynarodowych zespołach, zarówno pisemnie, jak i ustnie.

3. Czytanie i Rozumienie Kodów Źródłowych:

Wiele projektów open-source, jak również wewnętrzne projekty firm, wykorzystuje angielskojęzyczne nazewnictwo w kodzie – zmienne, funkcje i klasy są nazywane po angielsku. Znajomość angielskiego ułatwia zrozumienie takich projektów oraz przyspiesza wdrażanie się w nowe środowiska kodu.



Oto dlaczego znajomość angielskiego jest tak ważna dla współczesnego programisty:

4. Rozwiązywanie Problemów i Wsparcie Społeczności:

Społeczności programistów, takie jak Stack Overflow, GitHub czy Reddit, to ogromne źródła wiedzy i pomocy technicznej. Zdecydowana większość dyskusji odbywa się po angielsku, co oznacza, że biegłość w tym języku ułatwia zadawanie pytań i korzystanie z odpowiedzi innych programistów.

5. Uczenie się Nowych Technologii i Trendów:

Nowinki technologiczne oraz branżowe raporty (np. publikacje Google, Microsoft, GitHub) pojawiają się głównie w języku angielskim. Programiści, którzy znają angielski, są na bieżąco z nowymi technologiami i trendami, co pozwala im szybciej reagować na zmieniające się wymagania rynku.

6. Rozwój Kariery i Udział w Konferencjach:

Znajomość angielskiego otwiera drzwi do kariery na arenie międzynarodowej, umożliwia udział w konferencjach branżowych, szkoleniach i certyfikatach, które często są prowadzone właśnie w tym języku.



Wytyczne dotyczące projektu semestralnego

Studenci będą pracować nad **semestralnym projektem** w 1-2 osoby, stosując zasady OOP w języku C++ lub Java.

Projekty powinny demonstrować rozwiązywanie rzeczywistych problemów z odpowiednim wykorzystaniem klas, dziedziczenia, polimorfizmu, obsługi wyjątków, wzorców projektowych itp.

Zasady organizacji projektu:

1. Milestone-Based Development:

Students will submit progress in 3 milestones:

Milestone 1: Initial design and class diagrams (due Week 6).

Milestone 2: Core functionality with inheritance and polymorphism (due Week 10).

Milestone 3: Complete system with design patterns and unit testing (due Week 14).

2. Wzajemna weryfikacja:

Po Milestone 2, studenci dokonają wzajemnej oceny projektu innego zespołu w celu przekazania informacji zwrotnych i sugestii.



Wytyczne dotyczące projektu semestralnego

3. Prezentacja końcowa:

W 15/16 tygodniu studenci zaprezentują swój projekt. Zademonstrują swój kod, wyjaśnią decyzje projektowe i podkreślą użyte funkcje OOP.

Kryteria oceny :

Jakość kodu: Przestrzeganie zasad OOP (hermetyzacja, dziedziczenie itp.).

Projekt: Właściwe wykorzystanie wzorców projektowych.

Funkcjonalność: Jak dobrze projekt spełnia zamierzone cele.

Dokumentacja i testowanie: Przejrzysta dokumentacja kodu i dokładne testy jednostkowe.



Possible Project Topics (20 Ideas):

- Hospital Management System:** A system for managing patients, doctors, appointments, and treatments.
- Library Management System:** Track books, members, and borrowing history.
- Online Course Management Platform:** Similar to an LMS, for managing courses, students, and grades.
- ATM Simulation:** Simulate an ATM machine with authentication, balance inquiry, deposits, and withdrawals.
- E-commerce Store:** Implement a simple store with product browsing, cart management, and order processing.
- Bank Account System:** Manage multiple accounts, transactions, and generate account statements.
- Flight Reservation System:** Handle booking, cancellations, and availability for flights.
- Inventory Management System:** For tracking products, suppliers, and sales in a store.
- Real Estate Management System:** Manage property listings, buyers, and sales processes.
- University Management System:** Handle departments, professors, courses, and student enrollments.
- Hotel Reservation System:** Allow booking, cancellations, and managing room availability.
- Social Media App:** A simple platform for users to post messages, comment, and follow others.
- Smart Home Automation:** Control devices like lights, fans, and locks remotely.
- Taxi Dispatch System:** Manage taxi bookings, drivers, and passengers.
- Weather Forecasting System:** Fetch and display weather data from an API.
- Task Management Application:** Create, update, and track tasks for multiple users.
- Movie Ticket Booking System:** Allow users to book tickets, view showtimes, and choose seats.
- Chat Application:** A simple messaging app with multiple users and real-time updates.
- Vehicle Rental System:** Manage vehicle availability, rentals, and customer accounts.
- School Grading System:** For teachers to input grades, track student progress, and generate reports.



Dlaczego GitHub Classroom jest dobrą opcją?

Zalety usługi GitHub Classroom:

1. Przepływ pracy w świecie rzeczywistym:

GitHub jest standardem branżowym, więc studenci zdobywają praktyczne doświadczenie z narzędziami i przepływanymi pracą (pull requesty, zgłoszenia, kontrola wersji), których prawdopodobnie będą używać w środowiskach zawodowych.

Studenci uczą się współpracować za pośrednictwem **Git**, dzięki czemu świetnie nadaje się do projektów zespołowych.

2. Kontrola wersji i przejrzystość:

Praca każdego studenta jest śledzona w czasie, co pozwala zobaczyć jego postępy, zarządzać kodem i rozwiązywać problemy.

Możesz zidentyfikować, kiedy i gdzie uczeń wprowadził zmiany, co jest pomocne przy ocenianiu i przekazywaniu informacji zwrotnych.

3. Przegląd kodu i narzędzia opinii:

Wbudowana funkcja przeglądu kodu (za pośrednictwem **pull requestów i zgłoszeń**) umożliwia przekazywanie konkretnych opinii na temat poszczególnych linii kodu.

GitHub umożliwia również **wzajemną weryfikację**, promując współpracę studentów i pogłębianie wiedzy.



Dlaczego GitHub Classroom jest dobrą opcją?

Zalety usługi GitHub Classroom:

4. Automatyzacja zadań:

GitHub Classroom automatycznie dystrybuuje repozytoria do studentów.

Możesz skonfigurować **automatyczne przydzielanie** zadań za pomocą **GitHub Actions**, co ułatwia przekazywanie natychmiastowych informacji zwrotnych na temat zadań związanych z kodowaniem.

5. Integracja z innymi narzędziami:

GitHub Classroom można zintegrować z narzędziami takimi jak **Travis CI**, **CircleCI** i **GitHub Actions**, aby zautomatyzować testowanie i ocenianie.



POZNAN UNIVERSITY OF TECHNOLOGY

PROGRAMOWANIE OBIEKTOWE
WSTĘP DO PROGRAMOWANIA OBIEKTOWEGO
Serhii Baraban

PARADYGMATY PROGRAMOWANIA



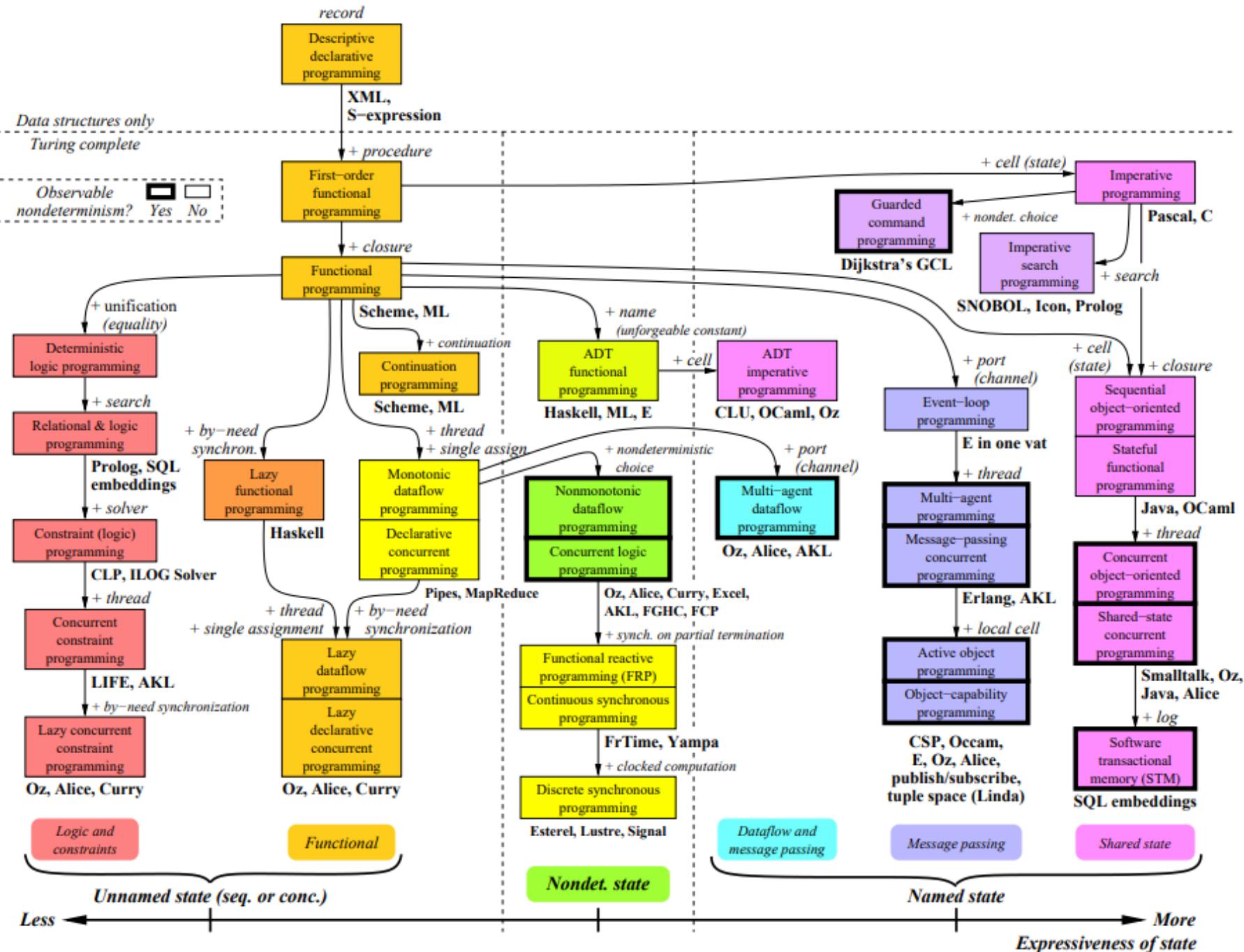
Czym są paradygmaty programowania?

Paradygmat programowania – styl lub sposób rozwiązywania problemów za pomocą kodu.

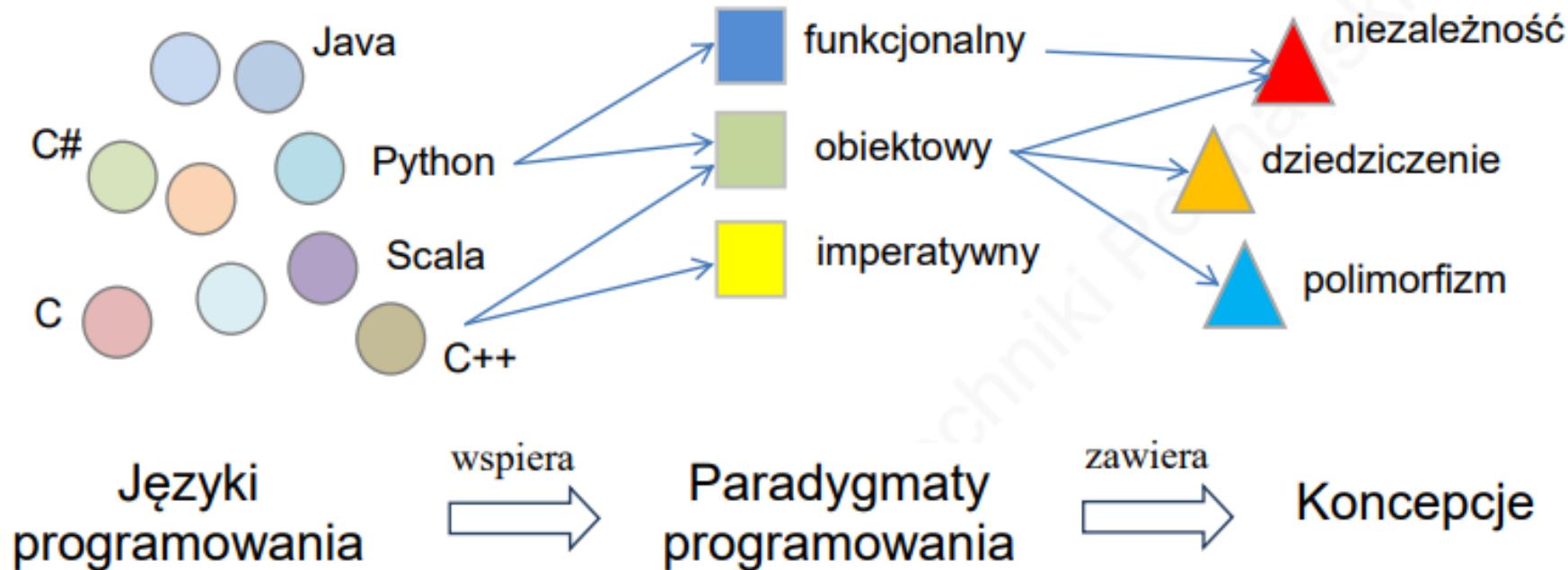
Określa **zasady i koncepcje**, które programista stosuje w celu tworzenia aplikacji.

Wpływa na sposób myślenia o problemie i na strukturę kodu.

Rysunek przedstawia wszystkich głównych paradygmatów programowania, zorganizowanych na wykresie, który pokazuje, w jaki sposób są one powiązane. Ten rysunek zawiera wiele informacji i wymaga dokładnego zbadania. Istnieje 27 pól, z których każde reprezentuje paradygmat jako zestaw koncepcji programowania.



PARADYGMATY I JĘZYKI PROGRAMOWANIA





Paradygmat Imperatywny

- **Podejście Imperatywne:** Skupia się na opisie krok po kroku, jak program ma osiągnąć cel.
- Programista określa **stan programu** i zmienia go za pomocą sekwencji instrukcji.
- Cechy:
 - Zmienne i przypisania wartości.
 - Instrukcje warunkowe (if, else) i pętle (for, while).
- Przykłady języków: **C, Python, Java**



Paradygmat Deklaratywny

- **Podejście Deklaratywne:** Opisuje, **co** ma zostać osiągnięte, a nie **jak** to zrobić.
- Programista definiuje oczekiwany wynik, a nie szczegółowe kroki.
- Cechy:
 - Często stosowany w bazach danych (SQL) i HTML.
 - Mniej kodu, łatwiejsze do zrozumienia.
- Przykłady języków: **SQL, HTML**



Paradygmat Programowania Obiektowego (OOP)

- **Programowanie Obiektowe:** Organizuje kod w postaci obiektów z własnymi atrybutami i metodami.

Kluczowe pojęcia:

Encapsulacja: Ukrywanie stanu obiektu i manipulacja przez metody.

Dziedziczenie: Reużycie kodu i rozszerzanie funkcjonalności.

Polimorfizm: Różne zachowania dla metod o tej samej nazwie.

- Przykłady języków: Java, C++, C#



Paradygmat Funkcyjny

- **Programowanie Funkcyjne:** Skupia się na **czystych funkcjach** i unika zmiennych stanów.

Cechy:

Funkcje czyste: Te same wejścia zawsze dają te same wyjścia, bez efektów ubocznych.

Rekurencja zamiast pętli.

Funkcje wyższego rzędu: Funkcje, które mogą przyjmować inne funkcje jako argumenty lub je zwracać.

- Przykłady języków: **Haskell, Lisp, JavaScript**



Paradygmat Logiczny

- **Programowanie Logiczne:** Opiera się na logice i regułach wyvodzenia wniosków.
- Skupia się na określeniu **zbioru faktów i reguł**, na podstawie których program wnioskuje.

Cechy:

Wyrażanie problemu przez fakty, reguły i zapytania.

Stosowane głównie w sztucznej inteligencji i przetwarzaniu języka naturalnego.

- Przykłady języków: **Prolog**



Zastosowania Paradygmatów

- **Imperatywny:** Systemy operacyjne, sterowniki, aplikacje z niskim poziomem abstrakcji.
- **Deklaratywny:** Bazy danych, interfejsy użytkownika, style CSS.
- **Obiektowy:** Systemy zarządzania, gry, aplikacje biznesowe.
- **Funkcyjny:** Przetwarzanie równoległe, analityka danych, przetwarzanie strumieniowe.
- **Logiczny:** Sztuczna inteligencja, systemy eksperckie, przetwarzanie języka.



Podsumowanie

Wybór paradygmatu zależy od rodzaju problemu i wymagań projektu.

Współczesne języki programowania często łączą **paradygmaty**:

Python: imperatywny, obiektowy, funkcyjny;

JavaScript: imperatywny, funkcyjny.

Świadomość paradygmatów pomaga w pisaniu elastycznego i łatwego w utrzymaniu kodu.



POZNAN UNIVERSITY OF TECHNOLOGY

PROGRAMOWANIE OBIEKTOWE
WSTĘP DO PROGRAMOWANIA OBIEKTOWEGO
Serhii Baraban

JEZYKI PROGRAMOWANIA



Czym jest język programowania?

Język programowania – formalny język służący do komunikacji z komputerem i definiowania poleceń, które komputer ma wykonać.

Składa się z zestawu **zasad składniowych i semantycznych**.

Programista pisze kod, który tłumaczony jest na język maszynowy za pomocą kompilatora lub interpretera.



Historia języków programowania – Lata 40. i 50.

Początki (lata 40. i 50.): Język maszynowy – pierwszy poziom języków, zapisany w kodzie binarnym (0 i 1).

Assembly – pierwszy złożony język niskopoziomowy, który pozwalał na bardziej czytelne instrukcje (stosowany w programowaniu pierwszych komputerów).

Fortran (1957) – pierwszy język wysokopoziomowy, przeznaczony do obliczeń naukowych i inżynierskich.



Historia języków programowania – Lata 60. i 70.

Lata 60.:

ALGOL – język algorytmiczny, który wprowadził strukturę blokową i stał się podstawą dla wielu języków.

COBOL – język stworzony z myślą o przetwarzaniu danych biznesowych.

Lata 70.:

C (1972) – uniwersalny język, który pozwalał na programowanie niskopoziomowe, stał się podstawą systemu UNIX.

Prolog – język logiczny, popularny w sztucznej inteligencji.

Pascal – język do nauki programowania, o czytelnej składni, wykorzystywany w edukacji.



Historia języków programowania – Lata 80. i 90.

Lata 80.:

C++ – rozszerzenie języka C o programowanie obiektowe, które umożliwiło lepszą organizację kodu i reużywalność.

Smalltalk – język obiektowy, który wprowadził czysto obiektowe podejście.

Lata 90.:

Python (1991) – język z prostą składnią, idealny do prototypowania i rozwoju aplikacji.

Java (1995) – język obiektowy, działający na różnych platformach dzięki maszynie wirtualnej Java (JVM).

JavaScript – język skryptowy stworzony do tworzenia dynamicznych stron internetowych, obecnie szeroko stosowany.



Typy języków programowania

Języki niskopoziomowe:

Działają blisko sprzętu, trudniejsze do nauki.

Przykłady: **Assembly, C.**

Języki wysokopoziomowe:

Bardziej zrozumiałe dla ludzi, ułatwiają tworzenie skomplikowanych programów.

Przykłady: **Python, Java, C++.**

Języki skryptowe:

Tworzone z myślą o szybkim rozwoju aplikacji, często dynamiczne.

Przykłady: **JavaScript, PHP, Ruby.**

Języki deklaratywne:

Koncentrują się na opisie, co ma być zrobione, a nie jak.

Przykłady: **SQL, HTML.**



Języki Kompilowane

Definicja: W językach komplikowanych kod źródłowy jest tłumaczony na język maszynowy za pomocą kompilatora przed uruchomieniem programu.

Proces Kompilacji: Tworzy plik wynikowy (np. .exe lub .o), który jest wykonywany bezpośrednio przez system operacyjny.

Wydajność: Wyższa, ponieważ kod jest przetwarzany raz podczas kompilacji, a następnie działa bez dodatkowych interpretacji.

Przykłady: C, C++, Java (kompilowana do bytecode na JVM).



Zalety języków kompilowanych:

Wysoka wydajność – Programy działają szybciej, ponieważ są przetwarzane na kod maszynowy.

Bezpieczeństwo typów – Języki kompilowane są zazwyczaj silnie typowane, co pomaga zapobiegać błędom w kodzie.

Optymalizacja – Kompilatory często optymalizują kod, aby działał szybciej lub zajmował mniej pamięci.



Języki Interpretowane

Definicja: Kod źródłowy jest przetwarzany przez interpreter podczas wykonania programu, linia po linii.

Proces Interpretacji: Interpreter tłumaczy kod na bieżąco, co sprawia, że nie jest potrzebny plik wynikowy.

Wydajność: Zazwyczaj niższa w porównaniu z językami kompilowanymi, ponieważ każda linia kodu jest tłumaczona na bieżąco.

Przykłady: Python, JavaScript, PHP.



Zalety języków interpretowanych:

Elastyczność – Kod można uruchamiać i testować bez konieczności komplikowania, co przyspiesza rozwój.

Platforma niezależna – Języki interpretowane działają na różnych systemach operacyjnych bez modyfikacji kodu.

Szybkie debugowanie – Programista może testować fragmenty kodu bez konieczności ponownego komplikowania całego programu.



Podsumowanie Różnic

Język	Kompilowany/Interpretowany	Typowanie	Typowe Zastosowanie	Paradygmaty
C	Kompilowany	Statyczne	Systemy wbudowane, programowanie systemowe	Imperatywny
C++	Kompilowany	Statyczne	Gry, aplikacje wysokiej wydajności	Obiektowy, strukturalny
Python	Interpretowany	Dynamiczne	Sztuczna inteligencja, analiza danych	Imperatywny, obiektowy, funkcyjny
JavaScript	Interpretowany	Dynamiczne	Aplikacje webowe, frontend i backend	Funkcyjny, obiektowy
Java	Kompilowany do bytecode	Statyczne	Aplikacje biznesowe, Android, systemy serwerowe	Obiektowy, wielowątkowy



OCENA JĘZYKÓW PROGRAMOWANIA

Informacje o organizacji **TIOBE**

Mierzymy jakość kodu oprogramowania.

TIOBE specjalizuje się w ocenie i śledzeniu jakości oprogramowania. Mierzymy jakość systemu oprogramowania, stosując powszechnie akceptowane metryki oprogramowania. Każdego dnia TIOBE sprawdza w czasie rzeczywistym ponad miliard linii kodu oprogramowania dla swoich klientów na całym świecie. Mierzymy jakość kodu oprogramowania w celu osiągnięcia świata wolnego od błędów poprzez ekspertyzę w zakresie jakości oprogramowania.

Firma TIOBE Software BV została założona 1 października 2000 roku przez Paula Jansena. Nazwa TIOBE to skrót od „The Importance Of Being Earnest”. Jest to również tytuł sztuki napisanej przez Oscara Wilde'a pod koniec XIX wieku. Wybierając tę nazwę, założyciele TIOBE podkreślają swoje szczerą i profesjonalne podejście do klientów, dostawców i współpracowników.

Większość pracowników TIOBE posiada tytuł magistra i/lub doktora informatyki. Stosujemy najnowocześniejsze metodologie i technologie, aby wspierać naszych klientów.

TIOBE Index for September 2023

Sep 2023	Sep 2022	Change	Programming Language	Ratings	Change
1	1		 Python	14.16%	-1.58%
2	2		 C	11.27%	-2.70%
3	4		 C++	10.65%	+0.90%
4	3		 Java	9.49%	-2.23%
5	5		 C#	7.31%	+2.42%
6	7		 JavaScript	3.30%	+0.48%
7	6		 Visual Basic	2.22%	-2.18%
8	10		 PHP	1.55%	-0.13%
9	8		 Assembly language	1.53%	-0.96%
10	9		 SQL	1.44%	-0.57%
11	15		 Fortran	1.28%	+0.26%

TIOBE Index for September 2023

11	15		 F	Fortran	1.28%	+0.26%
12	12		 GO	Go	1.19%	+0.03%
13	14		 A	MATLAB	1.19%	+0.13%
14	22		 S	Scratch	1.08%	+0.51%
15	13		 D	Delphi/Object Pascal	1.02%	-0.07%
16	16		 S	Swift	1.00%	+0.02%
17	26		 R	Rust	0.97%	+0.47%
18	18		 R	R	0.97%	+0.02%
19	20		 R	Ruby	0.95%	+0.30%
20	34		 K	Kotlin	0.90%	+0.59%



Top 10 programming languages in 2023

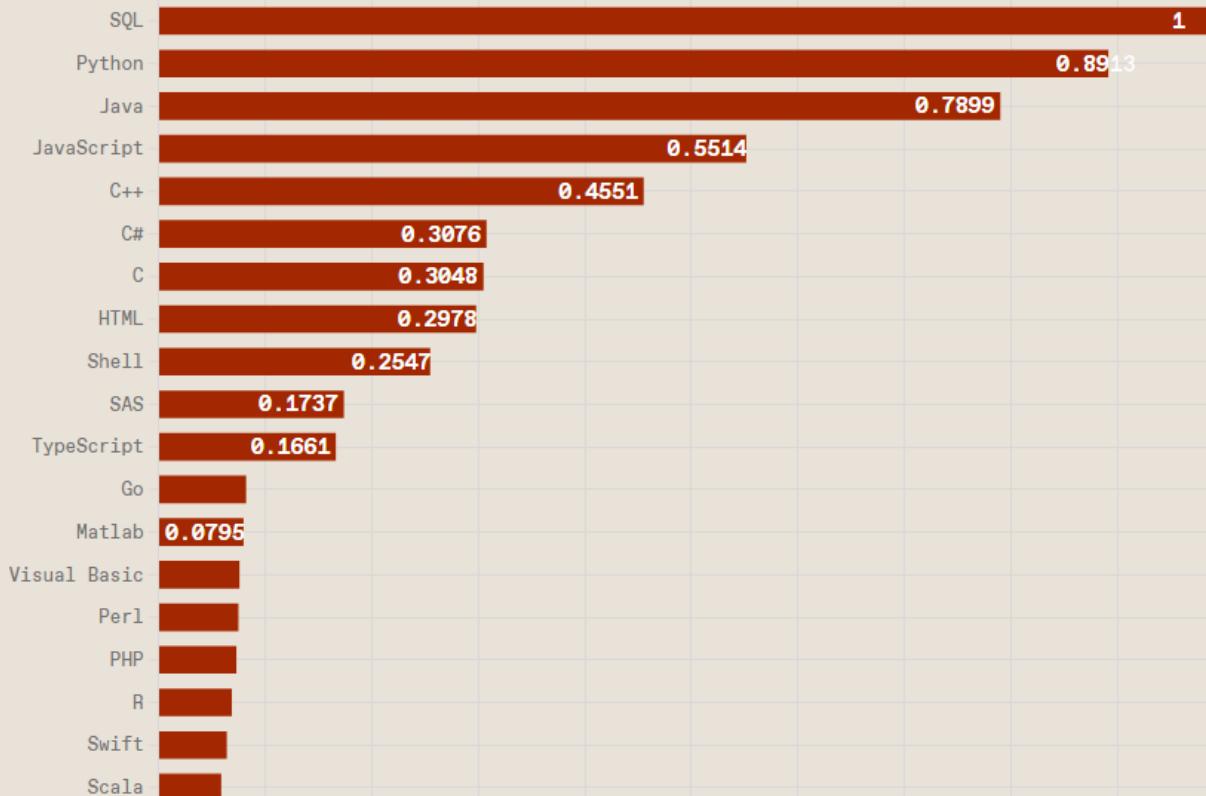
These are the most popular programming languages according to the PYPL index and Stack Overflow's Developer Survey 2023.

Position	PYPL ranking September 2023	Stack Overflow's Developer Survey 2023
#1	Python	JavaScript
#2	Java	HTML/CSS
#3	JavaScript	Python
#4	C#	SQL
#5	C/C++	TypeScript
#6	PHP	Bash/Shell
#7	R	Java
#8	TypeScript	C#
#9	Swift	C++
#10	Objective-C	C

Top Programming Languages 2023

Click a button to see a differently weighted ranking

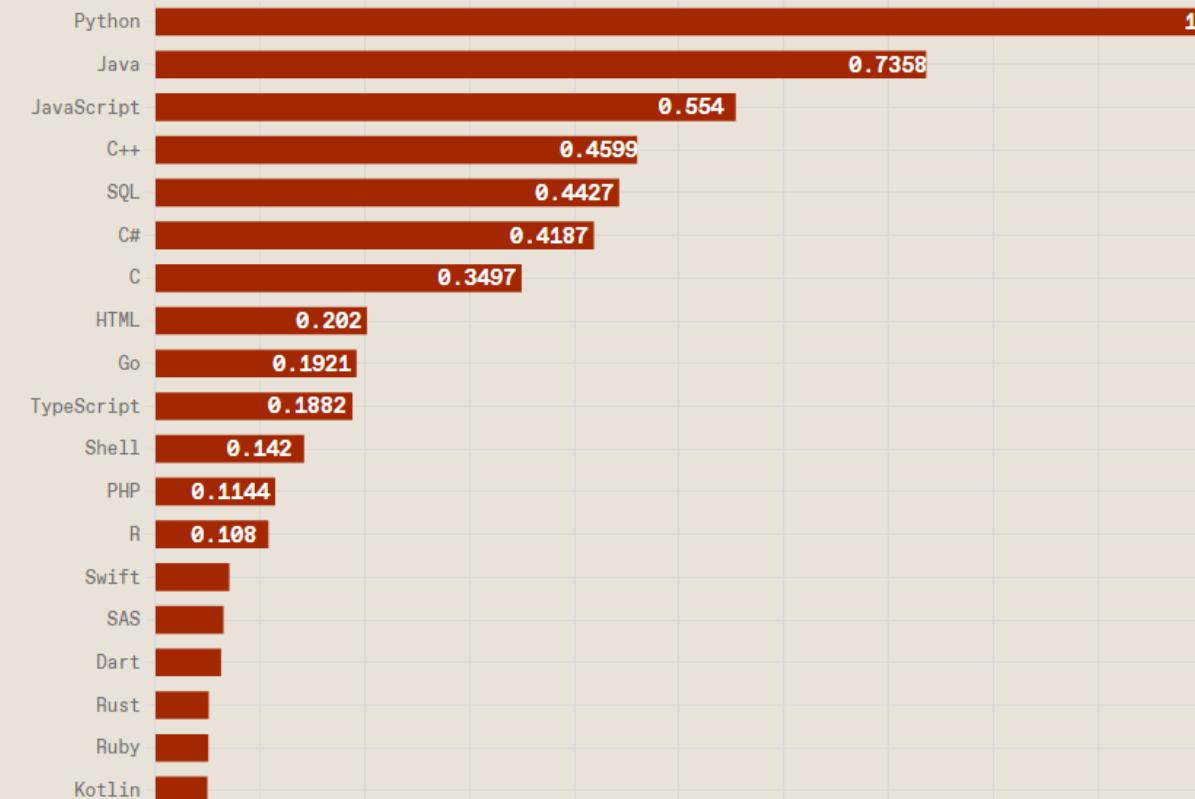
Spectrum **Jobs** Trending



Top Programming Languages 2023

Click a button to see a differently weighted ranking

Spectrum **Jobs** Trending





POZNAN UNIVERSITY OF TECHNOLOGY

PROGRAMOWANIE OBIEKTOWE
WSTĘP DO PROGRAMOWANIA OBIEKTOWEGO
Serhii Baraban

OBIEKTY I KLASY

Dlaczego potrzebujemy czegoś nowego?

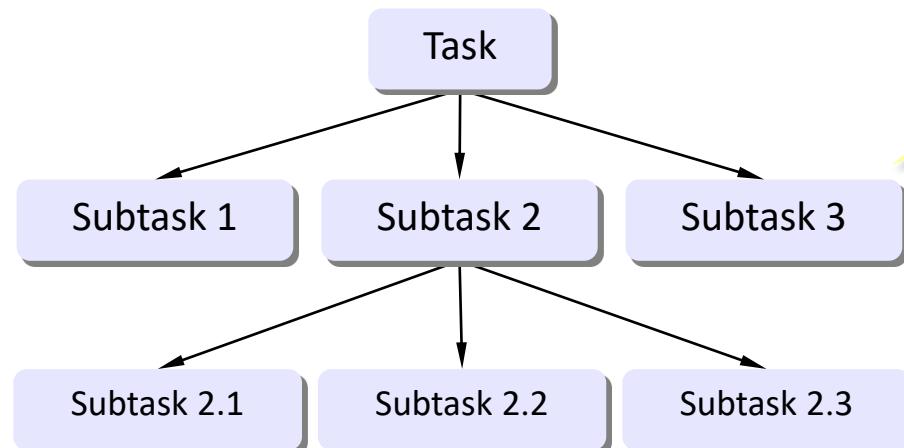


Główny problem - złożoność!

- Programy składające się z miliardów ciągów kodu
- -Tysiące zmiennych i tablic

Dejkstra: «W starożytności ludzie wynaleźli sposób na zarządzanie złożonymi systemami: „**dziel i rządź**”.».

Programowanie strukturalne:



Dekompozycja na zadania



osoba myśli inaczej,
przez **obiekty**



Dlaczego warto używać OOP?

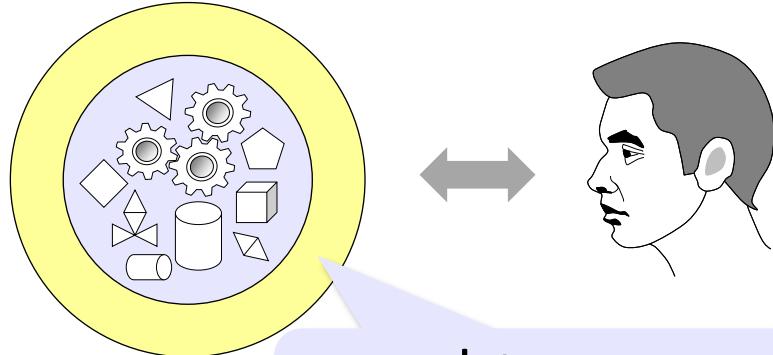
Programowanie obiektowe (OOP) jest jednym z najczęściej używanych paradymatów programowania.

Dlaczego jest powszechnie używany?

- Dobrze nadaje się do tworzenia trywialnych i złożonych aplikacji.
- Umożliwia ponowne wykorzystanie kodu, zwiększając tym samym produktywność.
- Nowe funkcje można łatwo wbudować w istniejący kod.
- Zmniejszone koszty produkcji i konserwacji.

Typowe języki programowania używane w OOP obejmują C++, Java, and C#

W jaki sposób postrzegamy obiekty?



podstawowe
właściwości

Abstrakcja to wybór podstawowych właściwości obiektu, które odróżniają go od innych obiektów.



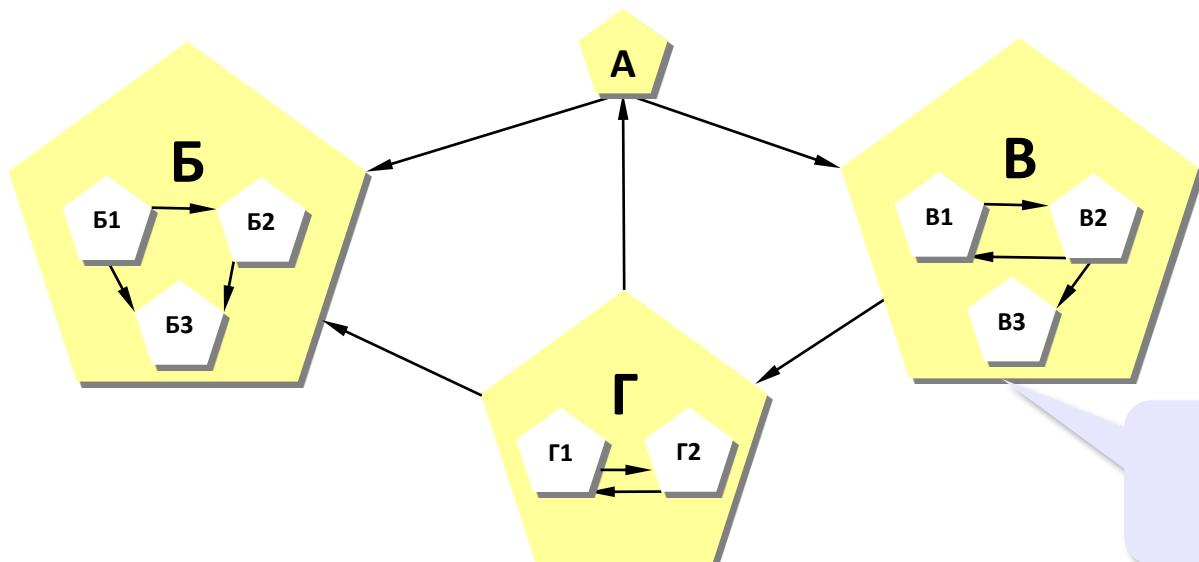
Różne cele - różne
modele!

Używanie obiektów

Program - zestaw obiektów (modeli), z których każdy ma własne właściwości i zachowanie, ale jego wewnętrzny układ jest ukryty przed innymi obiektami.



Konieczne jest „podzielenie” zadania na obiekty!



dekompozycja
według obiektów



Zaczynamy

Analiza obiektowa (OA):

- podświetlanie **obiektów**
- określić ich podstawowe **właściwości**
- opisać **zachowanie** (polecenia, które może wykonać)



Co jest obiektem?

Obiektem można nazwać coś, co ma wyraźne granice oraz posiada *stan* i *zachowanie*.

Stan określa zachowanie:

- osoba leżąca nie będzie skakać
- rozładowany pistolet nie wystrzeli

Klasa - jest to zbiór obiektów, które mają wspólną strukturę i wspólne zachowanie.



Komponenty OOP: Obiekty i klasy

Obiekt: modeluje

Obiekt świata rzeczywistego (np. komputer, książka, pudełko)

Pojęcie (np. spotkanie, rozmowa kwalifikacyjna)

Proces (np. sortowanie stosu papierów lub porównywanie dwóch komputerów w celu zmierzenia ich wydajności).

Klasa: prototyp lub schemat, na podstawie którego tworzone są obiekty.



Building Blocks of OOP: Objects & Classes

Klasa ma:

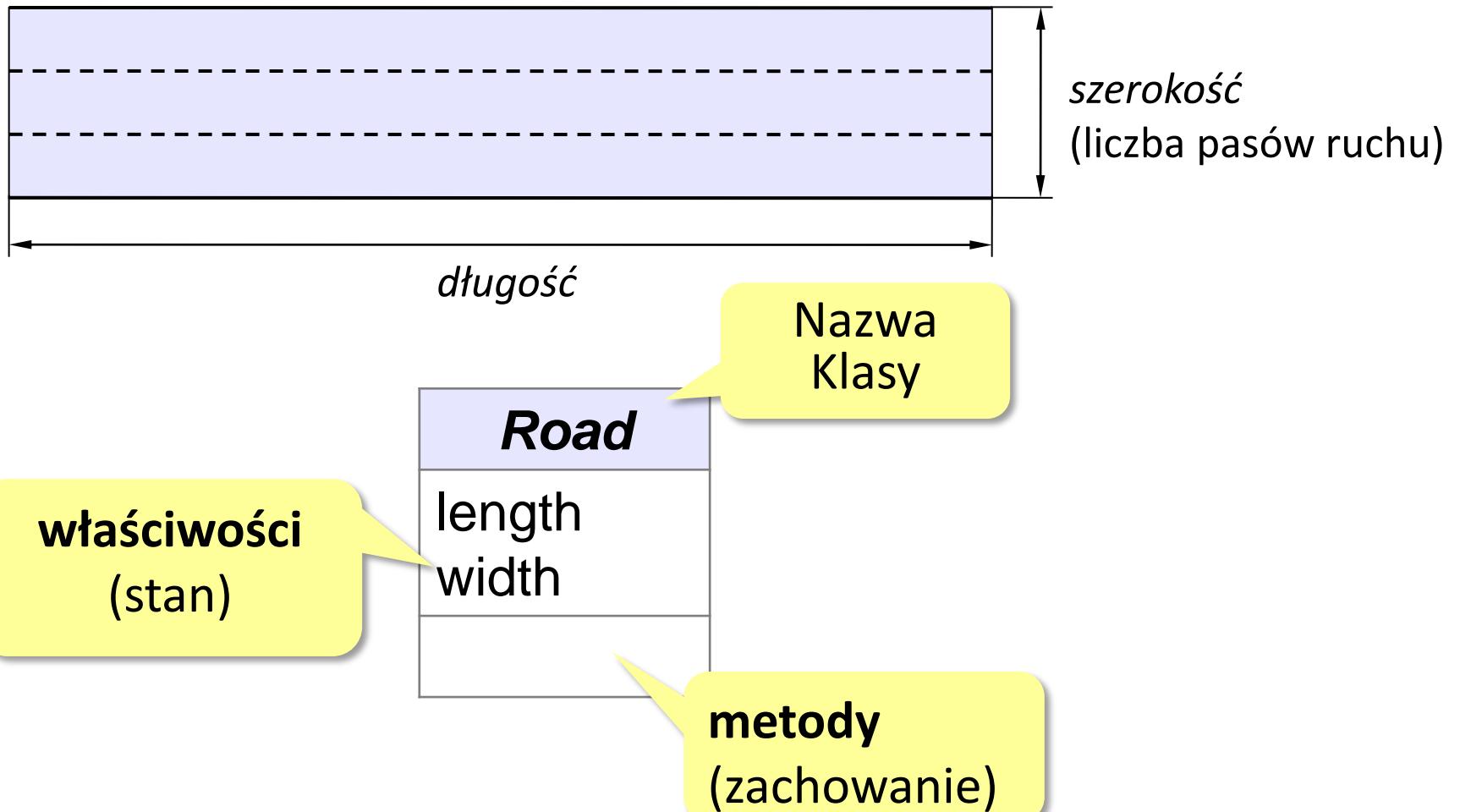
zestaw atrybutów lub właściwości, które opisują każdy obiekt
zestaw zachowań lub akcji, które każdy obiekt może wykonać

Obiekt ma:

zestaw danych (wartość dla każdego atrybutu)
zestaw akcji, które może wykonać
tożsamość

Model drogi z samochodami

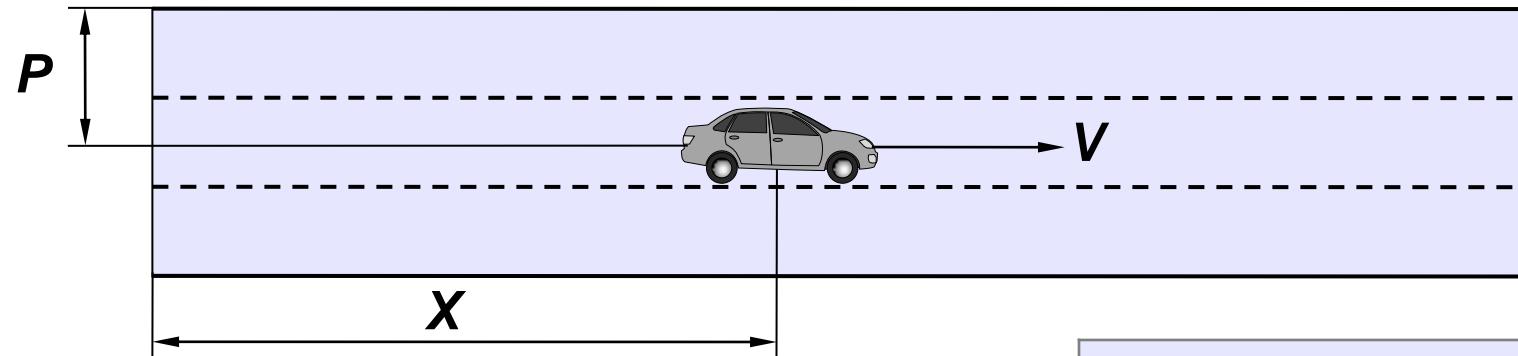
Obiekt „Droga”:



Road model with cars

Object «Auto»:

właściwości: współrzędne i prędkość



- wszystkie samochody są takie same
- prędkość jest stała
- jeden samochód na każdym pasie ruchu
- jeśli samochód przekroczy prawą granicę drogi, nowy samochód pojawi się zamiast niego po lewej stronie.

Auto
X (coordinate)
P (lane)
V (speed)
move

Metoda - procedura lub funkcja należąca do klasy obiektów.

Model drogi z samochodami

Interakcja między obiektami:

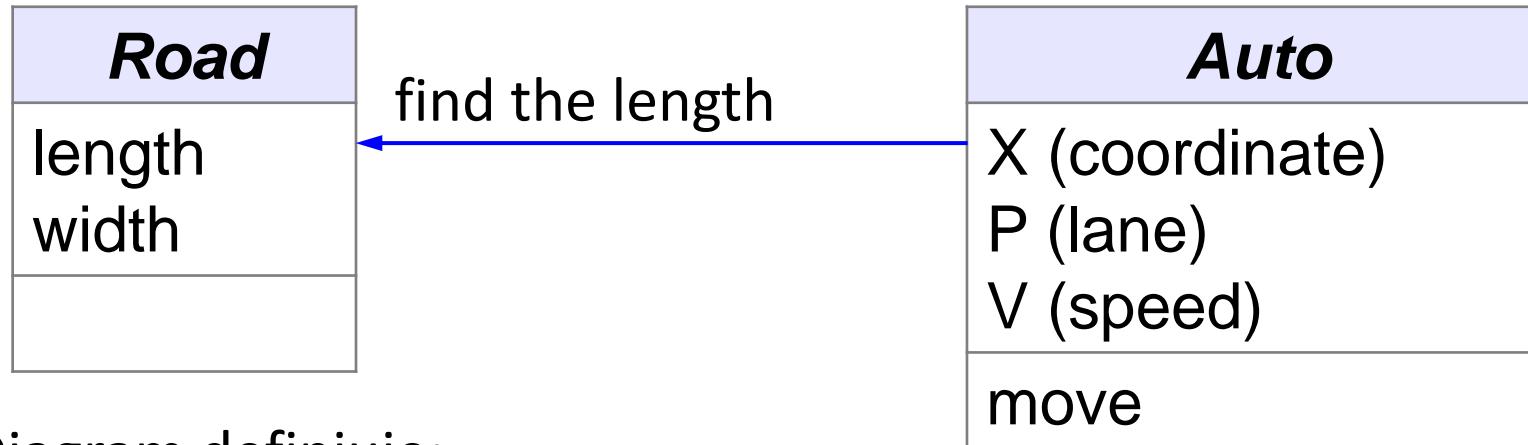


Diagram definiuje:

- **właściwości** obiektów
- **metody**: operacje, które może wykonać
- **połączenia** (wymiana danych) między obiektami



Ani słowa o wewnętrznym układzie obiektów!



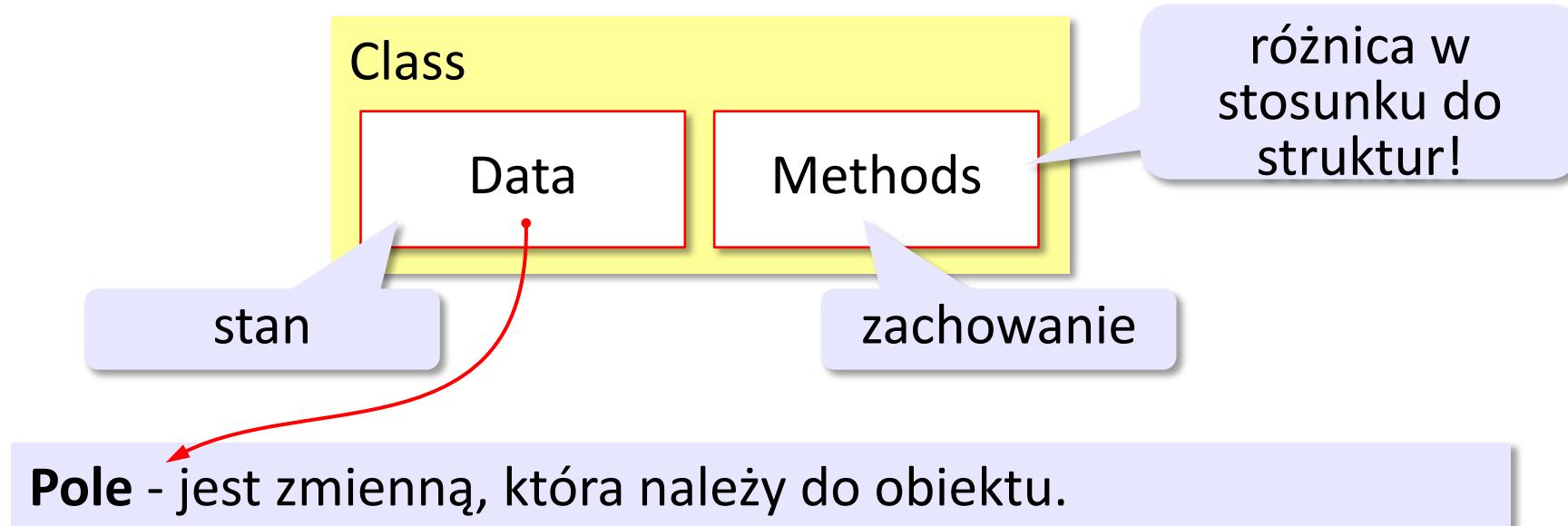
POZNAN UNIVERSITY OF TECHNOLOGY

PROGRAMOWANIE OBIEKTOWE
WSTĘP DO PROGRAMOWANIA OBIEKTOWEGO
Serhii Baraban

TWORZENIE OBIEKTÓW W PROGRAMIE

Klasy

- program - zestaw interakcji między samymi **obiektami**
- każdy obiekt jest instancją jakiejś **klasy**
- **klasa** - opis grupy obiektów o wspólnej strukturze i zachowaniu





Klasa «Road»

Opis klasy:

```
class TRoad:  
    pass
```



Instancje obiektów nie są tworzone!

Tworzenie obiektu:

```
road = TRoad()
```

wywołanie konstruktora

Konstruktor - to metoda klasy, która jest wywoływana w celu utworzenia obiektu tej klasy.



Konstruktor jest domyślnie tworzony automatycznie!

Nowy konstruktor - dodawanie pól

```
class TRoad:  
    def __init__( self ):  
        self.length = 0  
        self.width = 0
```

rekord punktowy

link odsyłający do samego obiektu

oba pola są ustawione na zero



Konstruktor ustawia początkowe wartości pól!

```
road = TRoad()  
road.length = 60  
road.width = 3
```

zmiana wartości pól

Konstruktor z parametrami

```
class TRoad:  
    def __init__( self, length0, width0 ):  
        self.length = length0  
        self.width = width0
```

automatycznie

Wezwać:

```
road = TRoad( 60, 3 )
```



Nie ma ochrony przed nieprawidłowymi danymi wejściowymi!



Ochrona przed nieprawidłowymi danymi

```
class TRoad:  
    def __init__( self, length0, width0 ):  
        if length0>0:  
            self.length=length0  
        else:  
            self.length=0  
        if width0 > 0:  
            self.width=width0  
        else:  
            self.width=0
```

```
self.length = length0 if length0 > 0 else 0  
self.width = width0 if width0 > 0 else 0
```

Klasa «Auto»

```
class TCar:  
    def __init__( self, road0, p0, v0 ):  
        self.road=road0  
        self.P=p0  
        self.V=v0  
        self.X=0
```

droga, po której
się porusza

pas ruchu

prędkość

współrzędna

Klasa «Auto» – method move

```
class TCar:  
    def __init__( self, road0, p0, v0 ):  
        ...  
    def move( self ):  
        self.x+=self.v  
        if self.x>self.road.length:  
            self.x=0
```

moving for $\Delta t = 1$

if outside
the road

Płynny ruch :

$$X = X_0 + V \cdot \Delta t$$

$\Delta t = 1$ interwał
próbkowania

ruch w jednostce czasu

Główny program

```
road = TRoad( 65, 3 )
car = TCar( road, 1, 10 )

car.move()
print( "After 1 step:" )
print( car.x )

for i in range(10):
    car.move()
    print( car.X )
```

```
class TCar:
```

```
...
```

```
def move( self ):
    self.X+=self.V
    if self.X>self.road.length:
        self.X=0
```



Jakie będą dane wyjściowe?

10 10

20

30

40

50

60

0

10

20

30

40

osiągnął koniec
drogi



Zestaw samochodów

```
N = 3
cars = []
for i in range(N):
    cars.append( TCar(road, i+1, 2*(i+1)) )

for k in range(100): # 100 steps
    for i in range(N): # for each car
        cars[i].move()

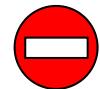
print( "After 100 steps:" )
for i in range(N):
    print( cars[i].x )
```

Korzyści i wady?

PO – to metoda tworzenia dużych programów!



- główny program jest prosty i przejrzysty
- klasy mogą być tworzone przez różnych programistów niezależnie od siebie (+interfejs!)
- powtórzenie użycia klas



- nieefektywny w przypadku małych zadań



Dziękuję za uwagę!