Sprawozdanie z laboratorium

Języki skryptowe

Py3 101: Wstęp do programowania

12.03.2025

Prowadzący: dr Beata Zieba

Grupa: 24-INF/A

Hubert Jarosz
Oliwer Pawelski

1 Wstęp

Lista zadań dotyczy różnych paradygmatów programowania oraz zagadnień związanych z językami programowania. Omawia różnice między imperatywnymi a deklaratywnymi językami, wskazując na sposób wyrażania logiki programów. Porusza również różnice między językami strukturalnymi i niestrukturalnymi, kładąc nacisk na organizację kodu. Dodatkowo opisuje języki obiektowe, ich wady i zalety, oraz istotność gramatyk i maszyn abstrakcyjnych w programowaniu. Zadania obejmują także języki interpretowane i kompilowane.

2 Zadania

Zadanie 1 **Opisać swoimi słowami różnicę pomiędzy imperatywnymi i deklaratywnymi językami programowania.**

Imperatywne język programowania polega na precyzyjnym określeniu jak należy wykonać określone zadanie. Programista zapisuje kolejne kroki, które komputer ma wykonać w określonej kolejności. Kod jest wykonywany sekwencyjnie. Opiera się na rozkazach. Przykłady języków imperatywnych:

- C++
- Java
- Python

Deklaratywne podejście polega na określeniu co ma zostać wykonane, bez podawania szczegółów dotyczących sposobu realizacji. Programista definiuje pożądany wynik, a sposób jego osiągnięcia pozostawia interpreterowi lub kompilatorowi. Przykłady języków deklaratywnych:

- SQL
- HTML
- Haskell

Zadanie 2 **Opisać różnicę pomiędzy strukturalnymi i niestrukturalnymi językami programowania.**

Strukturalne języki programowania organizują kod w funkcje, procedury i bloki, co poprawia jego czytelność i ułatwia zarządzanie. Wykorzystują pętle (for, while), instrukcje warunkowe (if-else) i unikają skoków (goto). Dzięki temu programy są bardziej przejrzyste, modułowe i łatwiejsze do debugowania. Przykłady to C, Python, Java i Pascal.

Niestrukturalne języki programowania nie wymuszają podziału kodu na funkcje, a sterowanie programem odbywa się głównie za pomocą skoków (goto). Kod jest często liniowy i trudniejszy w utrzymaniu. Programowanie w nich jest mniej czytelne i bardziej podatne na błędy. Przykłady to wczesne wersje BASIC, Assembly i stare języki jak COBOL.

Zadanie 3 **Opisać pojęcie języka obiektowego, opisać wady i zalety takiego paradygmatu programowania.**

Język obiektowy to język programowania, który opiera się na paradygmacie obiektowym (Object-Oriented Programming), gdzie programy są budowane wokół obiektów. Obiekt to jednostka łącząca dane (pola, atrybuty) oraz zachowania (metody). Kluczowe koncepcje OOP to klasy, dziedziczenie, enkapsulacja i polimorfizm, które pozwalają na lepsze modelowanie rzeczywistych systemów. Przykłady języków obiektowych to Java, C++, Python.

Zalety programowania obiektowego:

- Przejrzystość kodu (kod jest podzielony na klasy i obiekty, co ułatwia organizację i ponowne wykorzystanie.)
- Enkapsulacja (ogranicza bezpośredni dostęp do pól klasy, dzięki czemu dane są chronione przed nieautoryzowaną modyfikacją.)
- Lepsze odwzorowanie rzeczywistości (obiekty mogą reprezentować rzeczywiste byty, co ułatwia modelowanie systemów.)

Wady programowania obiektowego:

- Większe zużycie pamięci i wydajności
- Wydajność
- Jest trudniejsze niż programowanie strukturalne

```python
class Zwierze:
    def sound(self):
        return "Dzwiek"

class Pies(Zwierze):
    def sound(self):
        return "Hau Hau"

moj_pies = Pies()
print(moj_pies.sound())  # Dziedziczenie w klasycznym modelu programowania obiektowego
```

Listing 1: Kod do zadania 3

Zadanie 4 **Opisać swoimi słowami istotność gramatyk i maszyn abstrakcyjnych w kontekście programowania.**

Gramatyki w programowaniu określają zasady składniowe języka, umożliwiając analizowanie poprawności kodu i jego interpretację. Są kluczowe dla kompilatorów i interpreterów, używają one tych reguł do tłumaczenia kodu na zrozumiałą dla maszyny formę. Maszyny abstrakcyjne, takie jak maszyna Turinga czy maszyny wirtualne, służą do modelowania sposobu wykonywania programów i analizowania obliczeń. Pozwalają one na zrozumienie, jak działa algorytm oraz jakie ma ograniczenia obliczeniowe.

Zadanie 5 **Krótko omówić dlaczego pojęcie funkcji częściowej jest istotne w kontekście języków programowania.**

Pojęcie funkcji częściowej jest istotne w kontekście języków programowania, ponieważ odnosi się do funkcji, które nie są zdefiniowane dla wszystkich możliwych wartości wejściowych. W praktyce oznacza to, że funkcja może zwracać błąd lub być niedefiniowana dla niektórych argumentów (np. dzielenie przez zero). Zrozumienie funkcji częściowej pozwala programistom lepiej radzić sobie z obsługą wyjątków, walidacją danych oraz stabilnością programów

```
1 from functools import partial
2
3 def multiply(x, y):
4     return x * y
5
6 double = partial(multiply, 2)  # Tworzymy funkcje czesciowa,
7  # gdzie x=2
8
9 print(double(5))  # Output: 10
```

Listing 2: Kod do zadania 5. Przygład częściowej funkcji w python

Zadanie 6 **Krótko omówić dlaczego języki najniższego poziomu w konwencjonalnych architekturach (RISC, CISC, EPIC) nie mogą być językami deklaratywnymi.**

Wymagają one sprecyzowanego określenia kroków wykonania (np. rejestrów, adresów pamięci, operacji arytmetycznych), oraz są silnie powiązane z konkretną implementacją sprzętu. Wysoka kontrola nad sprzętem, jaką oferują te języki, nie pasuje do abstrakcji deklaratywnych, które ograniczają się do opisywania intencji, a nie precyzyjnych operacji.

Zadanie 7 **Własnymi słowami określić różnicę pomiędzy klasą a prototypem.**

Klasa to szablon lub blueprint, który definiuje strukturę i zachowanie obiektów w językach obiektowych, takich jak Java, Python czy C++. Obiekty są tworzone na podstawie klasy, a dziedziczenie odbywa się poprzez tworzenie

nowych klas, które rozszerzają istniejące. W klasach dziedziczenie jest zorganizowane w hierarchię, gdzie klasy potomne dziedziczą właściwości i metody klas rodziców. Z kolei prototyp to podejście, w którym obiekty dziedziczą bezpośrednio z innych obiektów, bez potrzeby definiowania klasy. To podejście jest typowe dla języków opartych na prototypach, takich jak JavaScript, gdzie obiekt może być prototypem dla innych obiektów, a właściwości i metody są dziedziczone w sposób bardziej elastyczny.

Zadanie 8 **Jaka jest różnica pomiędzy językami interpretowanymi i kompilowanymi. Omówić wady i zalety obu rozwiązań.**

Języki interpretowane to takie, w których kod źródłowy jest analizowany i wykonywany przez interpreter linia po linii, bez potrzeby wcześniejszej kompilacji. W czasie wykonywania kodu interpreter tłumaczy go na bieżąco na kod maszynowy, co może spowolnić wykonanie programu. Przykłady to Python, JavaScript.

Języki kompilowane to takie, w których kod źródłowy jest przetwarzany przez kompilator na kod maszynowy (lub pośredni) przed jego uruchomieniem. Kompilator wykonuje całą pracę na początku, a wynikowy plik wykonywalny jest niezależny od źródłowego kodu, co oznacza, że program jest uruchamiany bez potrzeby kompilacji za każdym razem. Przykłady to C, C++.

Zalety języków kompilowanych:

- Wydajność
- Brak zależności od interpretera
- Optymalizacja

Wady języków kompilowanych:

- Każda zmiana w kodzie wymaga ponownej kompilacji przed uruchomieniem programu.
- Skomplikowany proces debugowania

Zalety języków interpretowanych:

- Szybszy rozwój i testowanie – Kod można uruchomić natychmiast po jego napisaniu, co przyspiesza proces testowania i iteracji.
- Większa elastyczność – Można łatwiej modyfikować i testować program bez konieczności ponownej kompilacji.
- Łatwiejsze debugowanie – Interpreter umożliwia szybkie lokalizowanie błędów, ponieważ analizuje kod linia po linii.

Wady języków interpretowanych:

- Mniejsza wydajność
- Zależność od środowiska
- Brak optymalizacji

Zadanie 9 **Omówić twierdzenie Dijkstry o szkodliwości instrukcji skoku oraz teorię Bohm'a–Jacopini'ego w kontekście języków strukturalnych.**

Twierdzenie Dijkstry mówi, że instrukcje skoku (np. goto) w programach mogą prowadzić do chaotycznej, trudnej do utrzymania struktury kodu. Dijkstra twierdził, że programy powinny być pisane w sposób sekwencyjny, warunkowy i iteracyjny, co poprawia czytelność, zrozumiałość i łatwość w utrzymaniu kodu. Instrukcje skoku sprawiają, że kontrola przepływu programu staje się nieprzewidywalna, co utrudnia analizowanie i debugowanie aplikacji. Dijkstra postulował, że należy unikać takich skoków i zastępować je bardziej przejrzystymi konstrukcjami, takimi jak pętle i instrukcje warunkowe.

Znaczenie w kontekście języków strukturalnych: Języki strukturalne wprowadziły zasady strukturalnego programowania, które zakładają, że kod powinien być rozbity na moduły, funkcje i bloki. Programy powinny mieć jednoznaczny przepływ sterowania, który można łatwo śledzić. Języki takie jak C, Pascal czy Python umożliwiają programowanie bez używania instrukcji goto, zastępując je pętlami i instrukcjami warunkowymi, które pozwalają na przejrzysty i logiczny przepływ sterowania.

Zadanie 10 **Wymienić podstawowe cechy języka Python w zakresie składni i stosowanych mechanizmów w porównaniu do innych języków skryptowych.**

Cechy języka python:

- Prosta i czytelna składnia
- Dynamiczne typowanie
- Rozbudowany zestaw bibliotek standardowych
- Wieloparadygmatowość
- Łatwość pisania skryptów i prototypów

Język Python wyróżnia się na tle innych języków skryptowych dzięki swojej prostocie, dynamicznemu typowaniu, wszechstronności oraz dużej bibliotece standardowej. Dzięki łatwości nauki i możliwościom integracji z innymi technologiami, Python jest szczególnie ceniony w dziedzinie analizy danych i nauki maszynowej. W porównaniu do języków takich jak Ruby czy JavaScript, Python ma bardziej jednolitą składnię i bogate wsparcie dla paradygmatów

programowania, co czyni go wyborem numer jeden w wielu zastosowaniach skryptowych.