INFORMATYKA

Python i wokół niego

Języki programowania komputerów

Skąd aż tyle języków programowania?

- Z powodu ewolucji programowania komputerów
  - np. programowanie strukturalne → programowanie obiektowe
- Z powodu specjalnych wymagań
  - np. język C dla systemów operacyjnych, Prolog dla relacji itd.
- Z powodu osobistych przekonań
  - programiści mają swoje „fobie”, preferencje, przyzwyczajenia itp.
- Niektóre rozwiązania „pasują” lepiej do określonych wymagań

Skąd aż tyle języków programowania?

- Łatwość wykorzystania
  - w tym łatwość uczenia i nauczenia się
- Łatwość zastosowania
  - łatwość napisania kompilatora/interpretatora
- Dostępność dobrych kompilatorów
  - np. kompilatory dla języka Fortran w latach 50 i 60
- Czynniki ekonomiczne, patronat
  - np. COBOL (ang. *common business-oriented language*) do zastosowań biznesowych

Tłumaczenie języka programowania

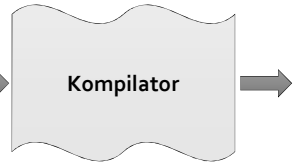
- Procesor języka programowania to dowolny system, który manipuluje programami zapisanymi w języku programowania
- Program źródłowy w jakimś języku źródłowym jest tłumaczony na obiekt programu w jakimś języku docelowym
- Tłumacze to asemblery lub kompilatory
- Asembler tłumaczy z języka „montażowego” na język maszynowy
- Kompilator tłumaczy z języka wysokiego poziomu na język niskiego poziomu
  - kompilator jest napisany w swoim języku implementacyjnym
- Interpreter to program, który akceptuje program źródłowy i uruchamia go (prawie) natychmiast
- Kompilator interpretacyjny tłumaczy program źródłowy na język pośredni, a wynikowy obiekt programowy jest następnie wykonywany przez interpreter

Kompilacja

- Proces zamiany tekstu z pliku ze źródłem programu (kodem) na kod maszynowy zrozumiały przez sprzęt komputerowy

```
#include <stdio.h>
int main(void)
{
  printf(”Hello”);
  return 0;
}
```

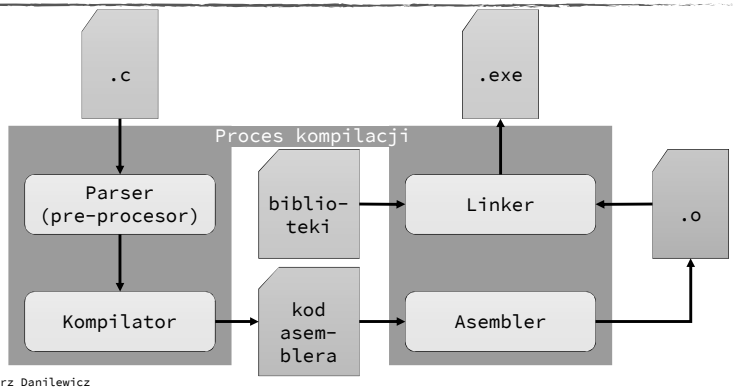
hello.c
Plik z kodem źródłowym




```
01001010
01010101
00001011
01011110
10010100
01001010
01010101
```

Kod wykonywalny
Kod maszynowy

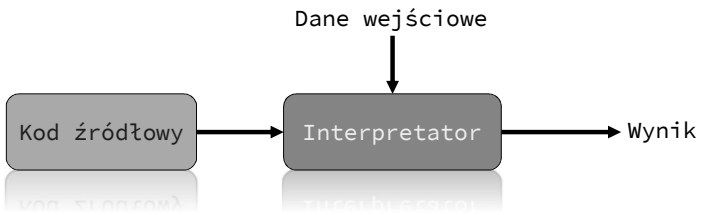
Proces kompilacji - fazy



Interpretacja

- Interpretator to procesor języka zaimplementowany w oprogramowaniu, który akceptuje dowolny program (program źródłowy) wyrażony w określonym języku (języku źródłowym) i natychmiast uruchamia ten program źródłowy
- Interpretator pobiera, analizuje i wykonuje instrukcje programu źródłowego, pojedynczo
- Program źródłowy zaczyna działać i generować wyniki, gdy tylko pierwsza instrukcja zostanie przeanalizowana
- Interpretator nie tłumaczy programu źródłowego na kod wynikowy przed wykonaniem
- Jednak:
  - faza analizy może obejmować lokalną translację na odpowiednią reprezentację pośrednią
  - interpretatory rekurencyjne mogą analizować cały program przed wykonaniem jakiejkolwiek instrukcji

Interpretator



Kompilacja a interpretacja

Kompilacja	Interpretacja
Dla programów produkcyjnych	Dla programów w fazie prototypowania/testowania/ odpluskwiania
Dla programów wykorzystywanych wielokrotnie	Dla programów uruchamianych „raz”
Dla skomplikowanych programów	Dla programów, dla których prędkość działania jest mniej istotna

Wady i zalety

Program kompilowany		Program interpretowany	
Zalety	Wady	Zalety	Wady
Gotowy do uruchomienia	Nieprzenaszalny	Przenaszalny	Wymaga interpretera
Często szybki	Nieelastyczny	Prosty do testowania	Często wolniejszy
Kod źródłowy jest prywatny	Wymaga dodatkowych działań	Łatwy do znajdowania błędów	Kod źródłowy ogólnie dostępny

Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od maszyny (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny (opcjonalnie)

Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od maszyny (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny (opcjonalnie)

Wynikiem działania kompilatora jest kod „montażowy” (assembly), który można wykonać za pomocą assemblera

Kompilacja i interpretacja

Działania niezależne od maszyny

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od maszyny (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny (opcjonalnie)

Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od maszyny (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny (opcjonalnie)

Wykrywanie symboli języka programowania – wykrywanie błędów językowych

Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od maszyny (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny (opcjonalnie)

Składanie symboli w struktury programowe – wykrywanie błędów składniowych (np. gdy symbole nie występują w oczekiwanej kolejności)

Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od maszyny (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny (opcjonalnie)

Sprawdzanie poprawności znaczeniowej w kodzie

Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od maszyny (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny (opcjonalnie)

```
cos = 5;
int cos;
```


Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od ma (opcjonalnie)

```
cos = 5;
int cos;
```

Obie linie kodu są składniowo poprawne (skaner i parser nie zgłoszą błędu)

Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od ma (opcjonalnie)

```
cos = 5;
int cos;
```

Jednak w języku C (jak i wielu innych) typ zmiennej musi być zadeklarowany przed użyciem zmiennej

Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od
  (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny
  (opcjonalnie)

```
cos = 5;
int cos;
```

Błąd znaczeniowy!

Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego od maszyny
  (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny
  (opcjonalnie)

Pozwala kompilatorom być w jakiejś części niezależnymi od maszyny

Kompilacja i interpretacja

- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośr
- Ulepszanie kodu niezależneg aszyny
  (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny
  (opcjonalnie)

Tłumaczenie kodu pośredniego w kod zależny od platformy i ewentualnie jego optymalizacja (np. w celu poprawy prędkości działania, użycia pamięci itp.)

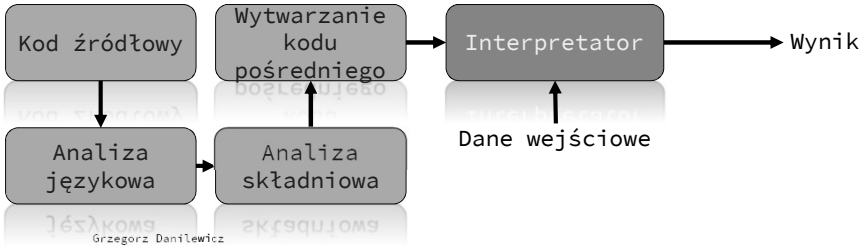
Kompilacja i interpretacja

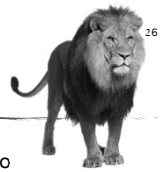
- Skanowanie
- Parsowanie
- Analiza znaczeniowa
- Wytwarzanie kodu pośredniego
- Ulepszanie kodu niezależnego maszyny
  (opcjonalnie)
- Wytwarzanie kodu docelowego
- Ulepszanie kodu zależnego od maszyny
  (opcjonalnie)

Kod wynikowy:
- Kod montażowy (assembly)
- Java bytecode (dla Javy)

Kompilacja i interpretacja

- Kompromis między szybszą kompilacją a rozsądnymi czasami wykonania

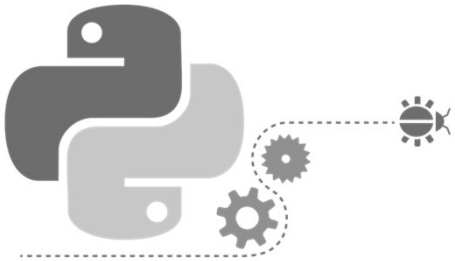




- Zaprojektowany tak, aby był intuicyjny i łatwy do pisania oprogramowania (bez poświęcania możliwości)
- Open source z dużą społecznością tworzącą pakiety i zasoby
- Jeden z najczęściej używanych języków programowania na świecie (obok C)
- Język „wypróbowany i działający”, który jest rozwijany od dłuższego czasu
- Wysokiej jakości wizualizacje
- Działa na większości systemów operacyjnych i platform – PRZENOSZALNOŚĆ!

- Wolniejszy niż programy skompilowane pisane w innych językach, takich jak C (duża część ekosystemu pythona napisana jest w C)
- Mniejsze/specjalistyczne pakiety mogą nie być dobrze przetestowane/utrzymywane

- Ze względu na swoje zalety oraz charakter pracy naukowej Python znajduje szerokie zastosowanie w nauce



Środowisko pracy Python

(propozycja)

Interpreter Python

- Wersje 3.x (wersja 2.7 nie jest już rozwijana)

  https://www.python.org/downloads/

  - dostępne dla wielu systemów operacyjnych (m.in. Windows, Linux/UNIX, macOS)
- Środowisko (pracy naukowej) Pythona – Anaconda

  https://www.anaconda.com/products/individual

  - dostępne dla wielu systemów operacyjnych (m.in. Windows, Linux/UNIX, macOS)
  - „wbudowane" IDE Spyder, „wbudowany" netebook Jupyter
- IDE

  https://code.visualstudio.com/

  - dostępne dla wielu systemów operacyjnych (m.in. Windows, Linux/UNIX, macOS)

ANACONDA

- DARMOWA! (Individual Edition)
- Pozwala tworzyć niezależne środowiska pracy z różnymi pakietami Pythona (np. gdy potrzebujemy konkretnej funkcjonalności)
- Przez nawigatora umożliwia szybkie uruchamianie niezbędnych narzędzi (na przykład notebook Jupyter lub – po zainstalowaniu – Visual Studio Code)
- Tworzy kompleksowe środowisko do uruchamiania skryptów napisanych w Pythonie

Visual Studio Code

Rekomenduję

- DARMOWE!
- Rozszerzalne – poprzez wtyczki (np. Python, C# i wiele wiele innych)
- Łatwo dostępny odpluskwiacz
- IntelliSense – podpowiedzi kontekstualne
- Wbudowana obsługa zarządzania kodem źródłowym (ang. *Source Code Management*) – git
- Integracja z chmurą Microsoft Azure