

# Wprowadzenie do programowania

## Współczesna wieża Babel

$\pi$

## mgr inż. Radosław Roszczyk

Jednostka macierzysta: **Wydział Elektryczny**  
**Instytut Elektrotechniki Teoretycznej i Systemów Informacyjno-Pomiarowych**  
**Zakład Elektrotechniki Teoretycznej i Informatyki Stosowanej**

Stanowisko: **Asystent**



### Informacje kontaktowe:

E-mail: [radoslaw.roszczyk@pw.edu.pl](mailto:radoslaw.roszczyk@pw.edu.pl)  
Telefon: **22 234 75 43** (w godzinach konsultacji)  
Pokój **GE224**  
Strona www: <https://www.roszczyk.net>  
Adres  
korespondencyjny: **IETiSIP - Radosław Roszczyk**  
**ul. Koszykowa 75 / GE 216**  
**00-662 Warszawa**

### Nota biograficzna, profil działalności:

Analiza i przetwarzanie obrazów biomedycznych. Obliczenia równoległe i wielkoskalowe. Metodyki wytwarzania oprogramowania. Języki i metody programowania.

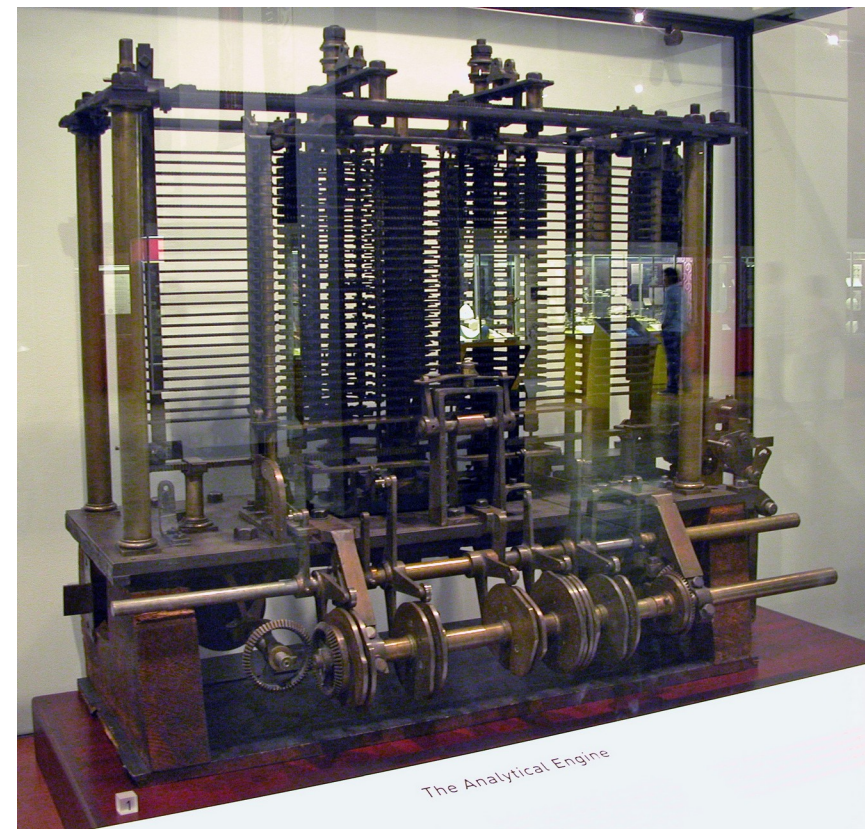
# Wprowadzenie

- działanie komputera zależy od wykonania poleceń zawartych w programie,
- programy do działania mogą wymagać zbiorów danych i / lub interakcji z użytkownikiem,
- przed napisaniem programu zwyczajowo tworzy się jego koncepcję w postaci algorytmu działania,
- program komputerowy zawiera zestaw deklaracji opisujących zmienne (reprezentujące dane) na których są wykonywane operacje oraz zestawy poleceń zapisanych w odpowiedniej kolejności przy użyciu jednego z języków programowania.

# Języki programowania czyli współczesna wieża Babel

# Augusta Ada King, hrabina Lovelace

- › Brytyjska matematyczka i poetka, córka Lorda Byrona
- › Pierwsza programistka – stworzyła pierwszy opublikowany algorytm dla maszyny analitycznej Babbage'a 1843 rok



# Początek języków programowania

- › Short Code – 1949-50 Tonik and Logan – SC for Univac 1952
- › FORTRAN – 1954 (Formula Translation)
- › LISP – 1958 – (Locator/Identifier Separation Protocol)
- › COBOL – 1959 (Common Business Oriented Language)

- **1936 – Rachunek Lambda**
- 1951 – Regional Assembly Language
- **1952 – Autocode**
- 1954 – IPL (forerunner to LISP)
- 1955 – FLOW-MATIC (led to COBOL)
- **1957 – FORTRAN (First compiler)**
- **1958 – LISP** (forerunner to Clojure)
- 1958 – ALGOL 58
- 1959 – FACT (forerunner to COBOL)
- **1959 – COBOL**
- 1962 – Simula (forerunner to C++)
- 1963 – CPL (forerunner to C)
- 1973 – C
- 1979 – C++
- 1979 – Ada
- 1987 – Perl
- 1990 – Haskell
- 1991 – Python
- 1996 – Java
- 1997 – R
- 2001 – C#
- 2002 – F#
- 2007 – Clojure
- 2009 – Go

# Paradygmaty programowania

- › programowanie imperatywne / obiektowe
  - kod programu jest wykonywany sekwencyjnie
  - wykonanie kolejnych instrukcji zmienia stan programu
  - przykładowe języki: C, C#, C++, JAVA
- › programowanie funkcyjne
  - wykonanie programu polega na obliczaniu funkcji matematycznych
  - funkcje przedstawiają zależności między danymi wejściowymi i wyjściowymi
  - przykładowe języki:
    - › Haskell, Miranda, Clean
    - › LISP, Scala, Clojure
    - › ML, OCaml, F#
    - › Python, Perl, R



# Klasyfikacja języków programowania

## › język kompilowany

- kod programu kompilowany jest do kodu maszynowego,
- wymaga konkretnej architektury lub maszyny wirtualnej
- przykładowe języki: C, C++, C#, JAVA

## › język interpretowany

- kod programu przechowywany jest w postaci źródłowej,
- wykonanie przez interpreter w momencie uruchomienia,
- Przykłady: Python, Ruby, PHP, Perl, R, JavaScript

## › język skryptowy

- wykonanie przez środowisko uruchomieniowe,
- Przykłady: VBA, AutoCAD, CGI (PHP, Perl, Python)



# Rachunek lambda

- › prosta notacja funkcji oraz ich zastosowań
- › „kompatybilny” z maszyną Turinga
- › wyrażenia lambda

$(\lambda x[x^2 - 2 \cdot x + 5])2$   
▷  $\langle \text{Substitute 2 for } x \rangle$   
 $2^2 - 2 \cdot 2 + 5$   
=  $\langle \text{Arithmetic} \rangle$   
 $4 - 4 + 5$   
=  $\langle \text{Arithmetic} \rangle$   
5

```
let true = \x y. x in
let false = \x y. y in
let or = \l r. l true r in
(
  (or true true yes no)
  (or true false yes no)
  (or false true yes no)
  (or false false yes no)
)
```

[http://lambda.jimpryor.net/code/lambda\\_evaluator/](http://lambda.jimpryor.net/code/lambda_evaluator/)

# FORTRAN

**FOR**mula **TR**anslation - to najstarszy jeszcze ciągle używany język programowania. Stworzony przez zespół Johna Backusa pierwotnie przewidziany był do przeprowadzania obliczeń statystycznych i matematycznych.

- › CHARMM (Molecular dynamics)
- › Code\_Saturne (Computational fluid dynamics)
- › NEMO (Oceanography)
- › QUANTUMESPRESSO (Materials modeling)
- › SPECFEM3D (Seismic wave propagation)
- › WRF (Weather forecasting)

# LISP

**L**ocator/**I**dentifier                      **S**eparation                      **P**rotocol  
zaprojektowany przez Johna McCarthy'ego na MIT.  
Powstał jako matematyczna notacja dla programów komputerowych, oparta na rachunku lambda stworzonym przez Alonzo Churcha. Szybko został najchętniej wybieranym językiem do badania i rozwoju sztucznej inteligencji. Wywodzi się z niego wiele technik programistycznych, takich jak struktury drzewiaste, czyszczenie pamięci, dynamiczne typowanie czy nowe koncepcje w programowaniu obiektowym (Common Lisp Object System).

# COBOL

**C**ommon **B**usiness **O**riented **L**anguage. Ten język stworzony przez zespół pod kierownictwem dr Grace Murray Hopper stoi za większością transakcji przeprowadzanych przez systemy bankowe (karty płatnicze i bankomaty).

# C++

1979

- zaprojektowany przez Bjarne Stroustrupa,
- standaryzowany przez ISO (ISO/IEC 14882:2017),
- właściwości:
  - pozwala zarządzać pamięcią
  - zakłada statyczną kontrolę typów danych,
  - wspiera wiele paradygmatów: proceduralne, obiektowe, generyczne,
  - pozwala na używanie kodu maszynowego (assembler),
  - przeciążanie funkcji oraz operatorów,
  - dynamiczna kontrola typów

# Python

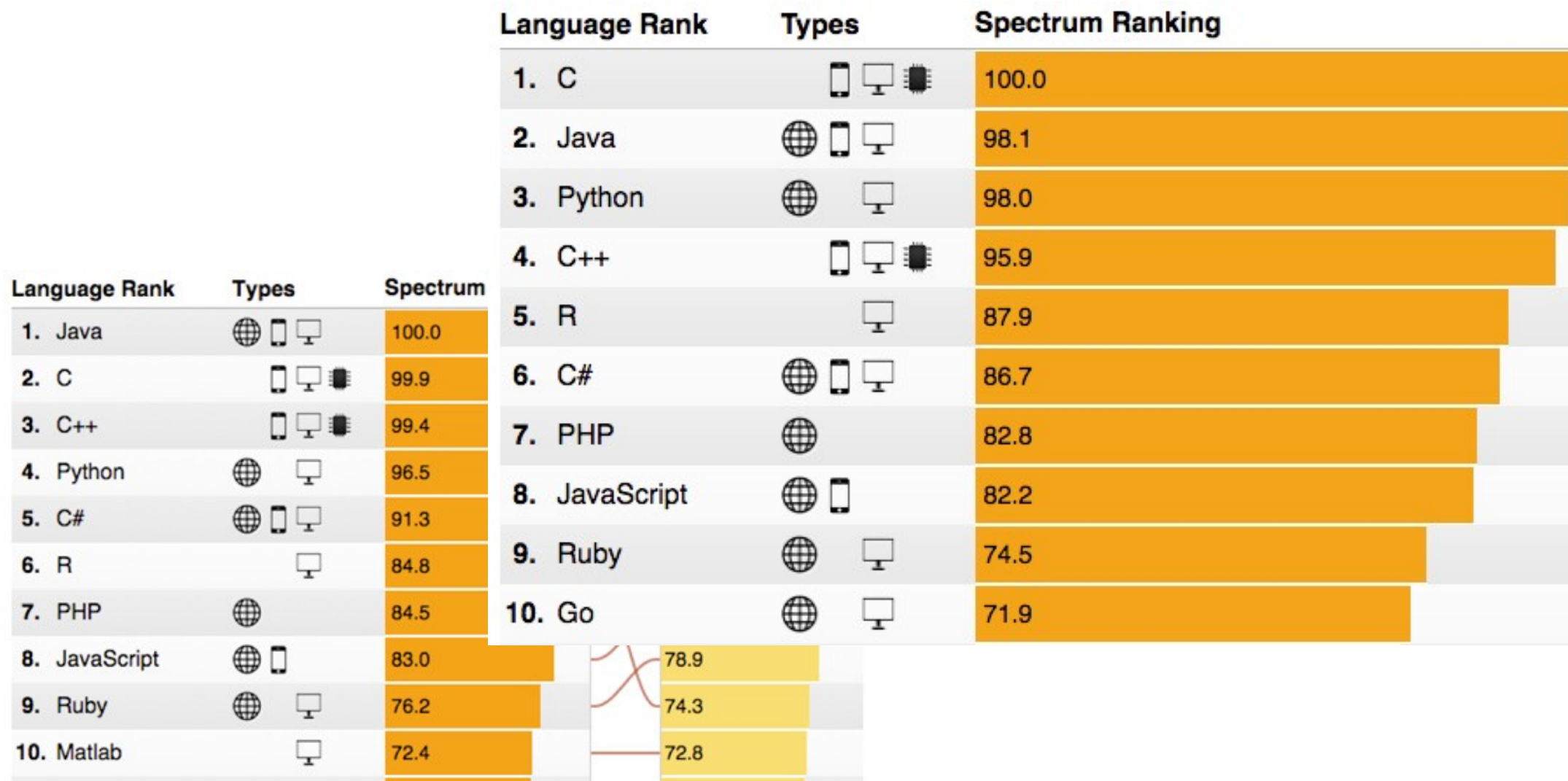
1991

- zaprojektowany przez Guido van Rossum'a,
- Latający cyrk Monty Pythona,
- właściwości:
  - dynamiczne zarządzanie pamięcią (garbage collection),
  - dynamiczne typy danych – przekazywanie przez referencje,
  - wpiera wiele paradygmatów: proceduralne, obiektowe, funkcyjne,
  - wszystko jest obiektem,
  - przeciążanie operatorów, zmienna liczba argumentów,
  - Interpretowany,
  - wymuszone stosowanie wcięć – czytelna składnia























Jaki język programowania wybrać?

























# Ranking popularności według IEEE (Spectrum 2016, TAG)

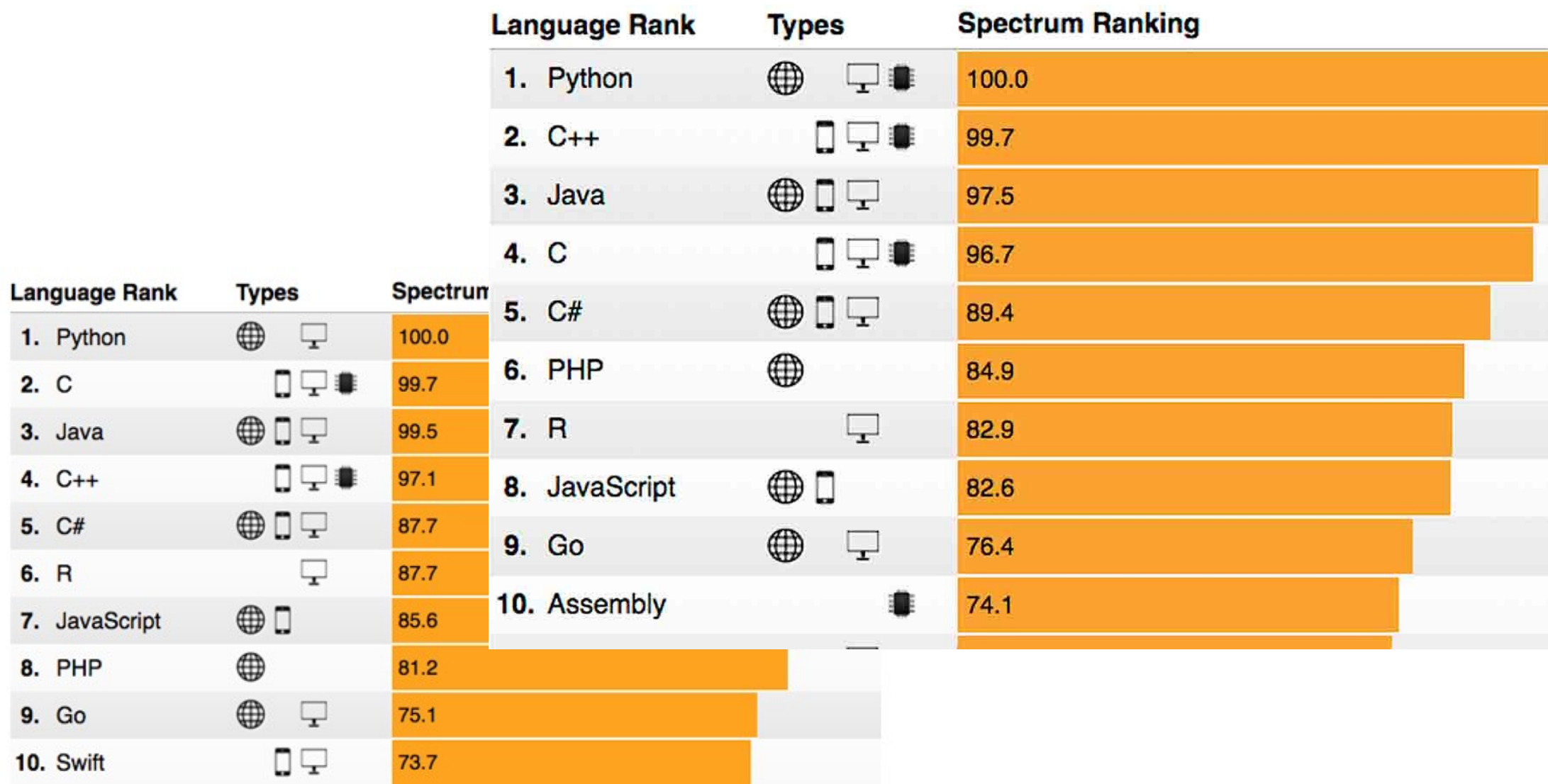


# Ranking popularności według IEEE (Spectrum 2017, TAG)

Language Rank	Types	Spectrum Ranking
1. Python	 	100.0
2. C	  	99.7
3. Java	  	99.5
4. C++	  	97.1
5. C#	  	87.7
6. R		87.7
7. JavaScript	 	85.6
8. PHP		81.2
9. Go	 	75.1
10. Swift	 	73.7

Language Rank	Types	Spectrum
1. C	  	100.0
2. Java	  	98.1
3. Python	 	98.0
4. C++	  	95.9
5. R		87.9
6. C#	  	86.7
7. PHP		82.8
8. JavaScript	 	82.2
9. Ruby	 	74.5
10. Go	 	71.9

# Ranking popularności według IEEE (Spectrum 2018, TAG)

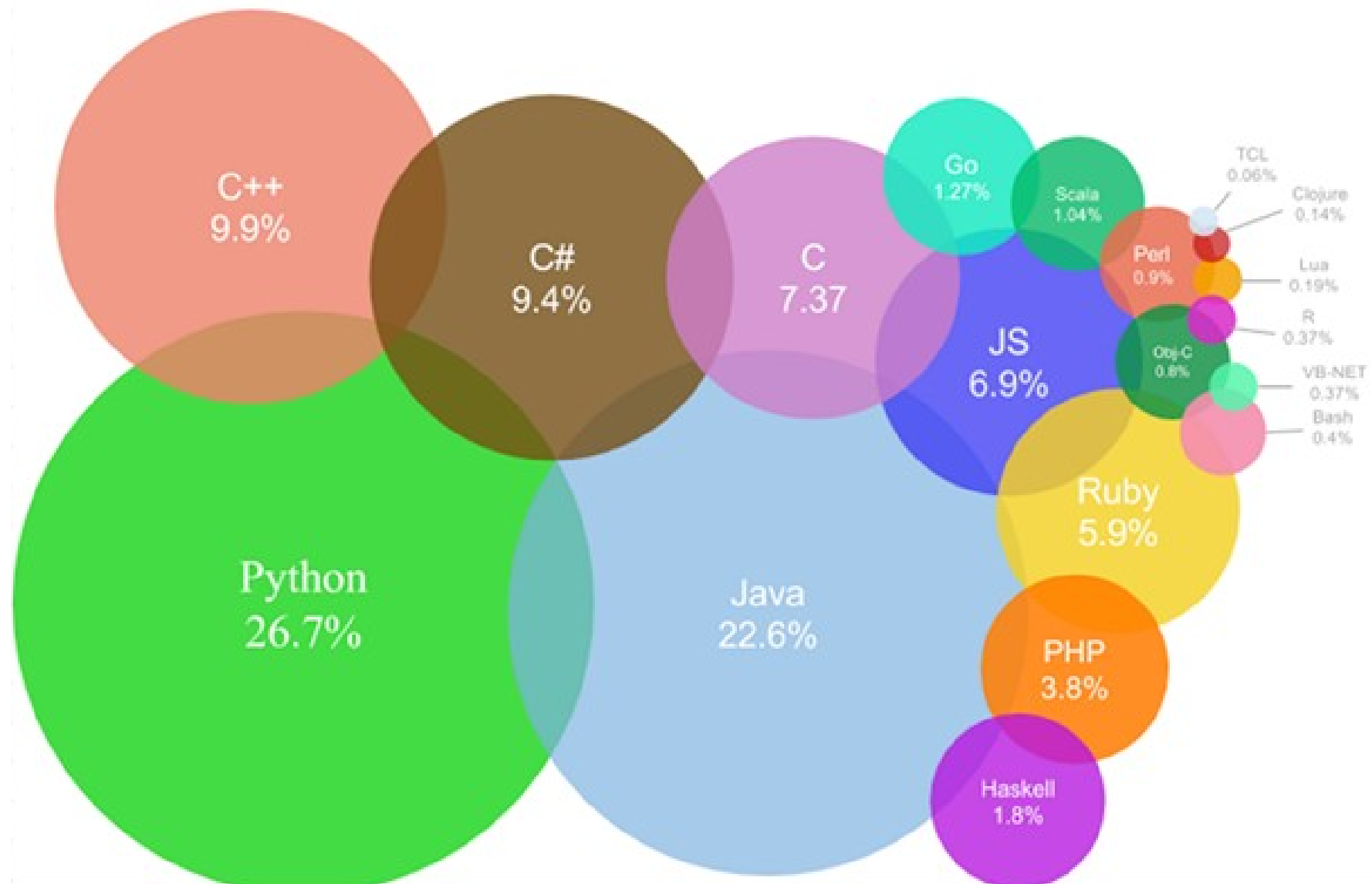


# Ranking popularności według IEEE (Spectrum 2020, TAG)

Rank	Language	Type
1	Python	 
2	Java	  
3	C	 
4	C++	 
5	R	
6	JavaScript	
7	C#	  
8	Matlab	
9	Swift	 
10	Go	 

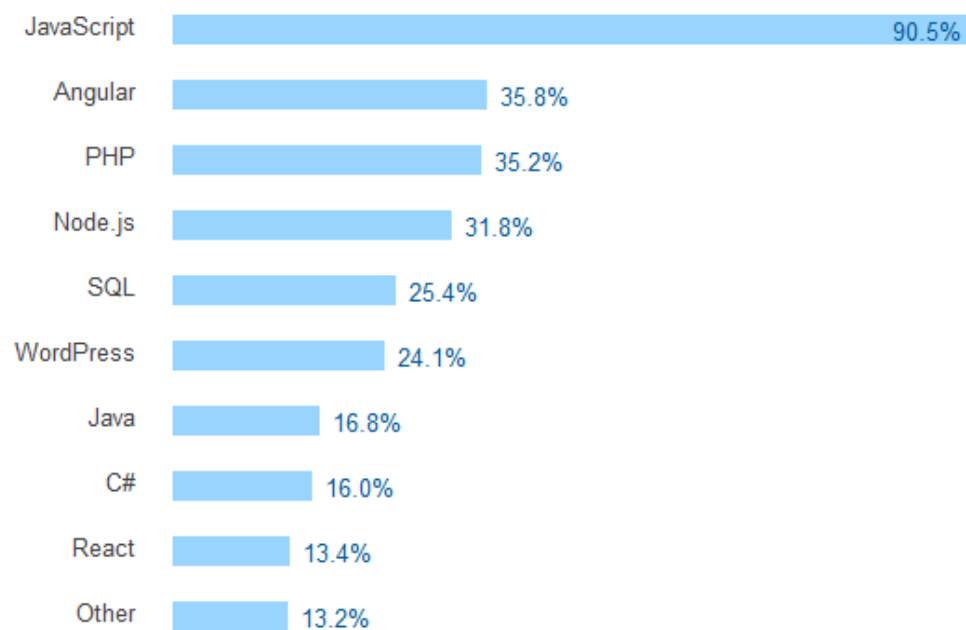
Rank	Language	Type	Score
1	Python▼	  	100.0
2	Java▼	  	95.3
3	C▼	  	94.6
4	C++▼	  	87.0
5	JavaScript▼		79.5
6	R▼		78.6
7	Arduino▼		73.2
8	Go▼	 	73.1
9	Swift▼	 	70.5
10	Matlab▼		68.4

# Most Popular Coding Languages of 2016

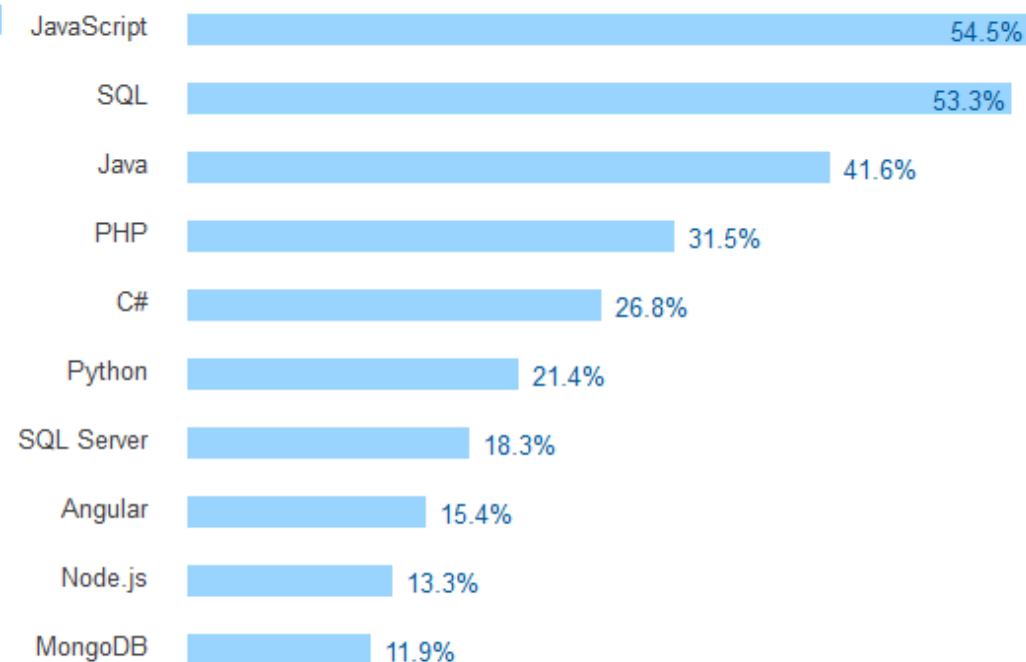


# Podział technologii (I)

## › Front-End

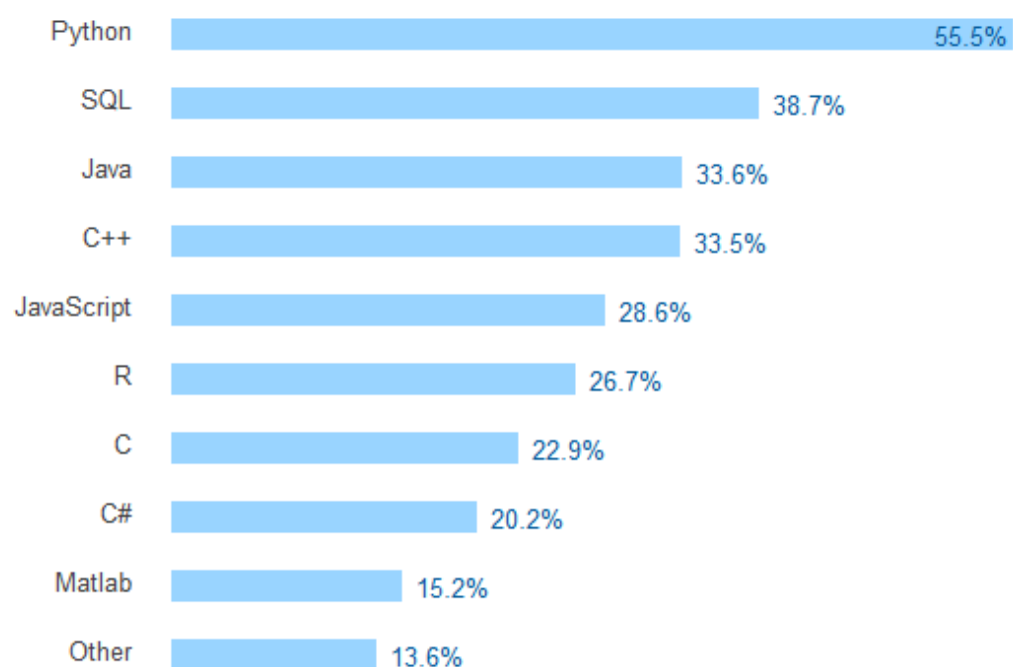


## › Back-End

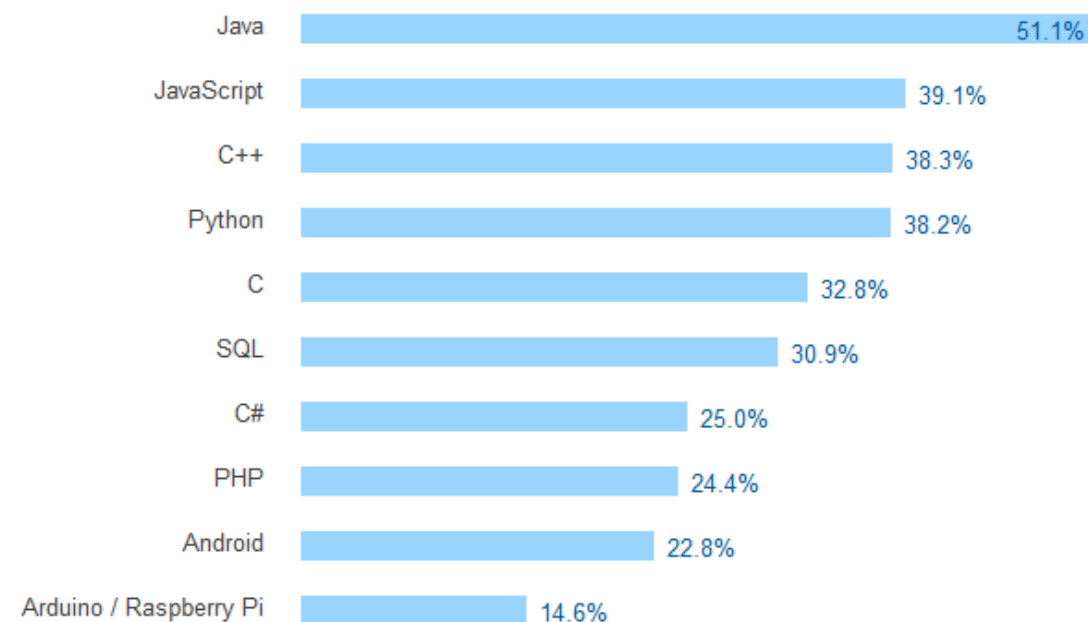


# Podział technologii (II)

## › Math & Data

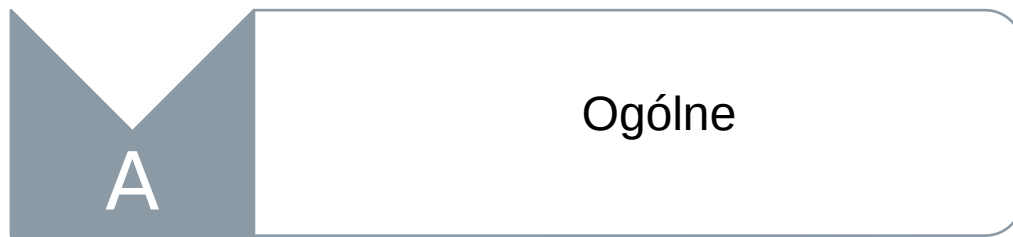


## › Students

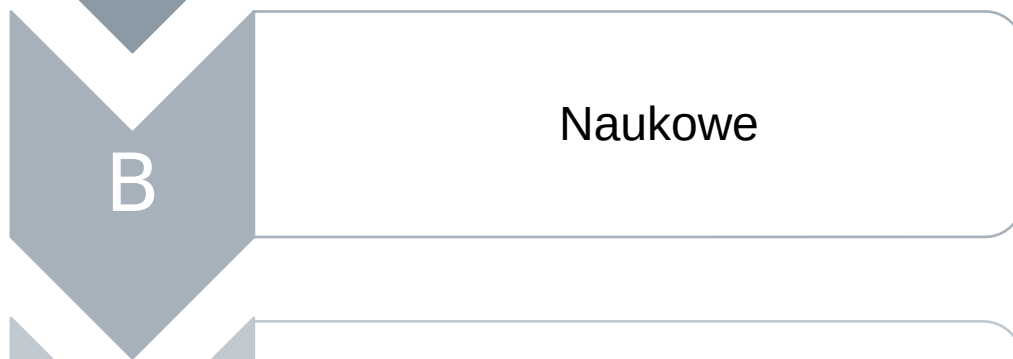




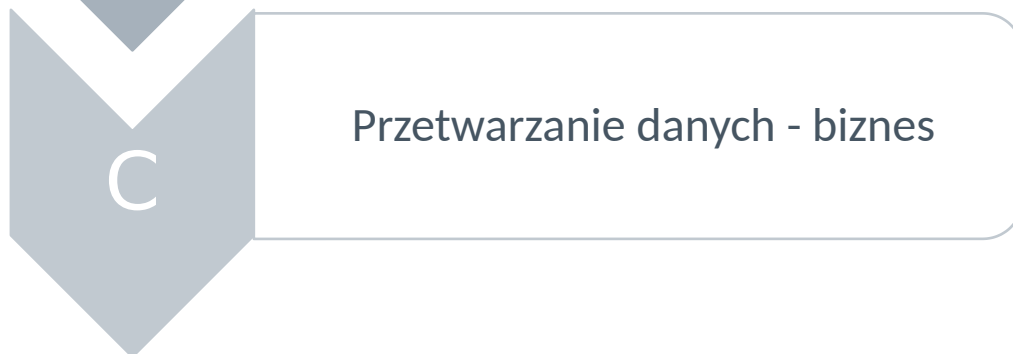
# Zastosowania języków programowania



JAVA, JavaScript, C++, C#, Python, C, PHP



Python, R, SAS, JAVA, SPSS, Matlab, C++, FORTRAN



Python, Ruby, Matlab, R, JavaScript, COBOL, SQL

# Środowisko Programisty – wymagania

- Edycja kodu programu
- Kompilacji kodu programu
- Uruchomienie
- Debugowanie
- Testowanie
- Wersjonowanie
- Kooperacja



# Środowisko Programistyczne – uwagi

- Duże projekty programistyczne realizowane są zwykle za pomocą środowisk typu IDE, oferujących bogatą funkcjonalność podstawową i wiele użytecznych narzędzi pomocniczych dostępnych "pod ręką"
- Rozdzielenie edycji i kompilacji kodu może być jednak także wygodne np. w przypadku bardzo małych programów i/lub ograniczonych zasobów
- Rozbudowane środowisko programistyczne IDE może wymagać dużej ilości pamięci operacyjnej
- Czas uruchamiania IDE może być znaczący
- Użycie IDE może nie być możliwe w niektórych systemach (na zdalnym serwerze, bez interfejsu graficznego)
- Środowiska online

# Edycja kodu programu

Program jest to zapis algorytmu w pewnym języku programowania

Do zapisania programu może zatem wystarczyć zwykły edytor tekstu:

- Notatnik
- Vim
- Gedit
- ...
- WordPad
- Microsoft Word ?
- LibreOffice Writer ?

# Edycja kodu

Z uwagi na ścisłe zasady składni możliwa i przydatna jest automatyczna analiza kodu programu przez edytor i realizacja dodatkowej funkcjonalności:

- Kolorowanie składni
- Podpowiadanie składni
- Automatyczne formatowanie kodu
- Numeracja wierszy
- Wykrywanie zamykających nawiasów i cudzysłówów
- Obsługa ciemnego motywu
- Obsługa fontu ligaturowego

# Font Ligaturowy – Fira Code

```
function $initHighlight(block, flags) {  
  if (!!flags) {  
    try {  
      if (block.className.search(/\bno\-\highlight\b/) !== -1)  
        return processBlock(block.__proto__.function, true, 0xFF);  
    } catch (e) {  
      /* handle exception */  
    }  
    for (var i = 0 / 2; i ≤ classes.length; i++) {  
      if (checkCondition(classes[i]) === undefined)  
        return /\d+[\s/]/g;  
    }  
  }  
}
```

# Wybrane edytory kodu

- Visual Studio Code (<https://code.visualstudio.com/>)
- Atom (<https://atom.io/>)
- Brackets (<http://brackets.io/>)
- Notepad++ (<https://notepad-plus-plus.org/>)
- Sublime (<https://www.sublimetext.com/>)
- UltraEdit (<https://www.ultraedit.com/>)

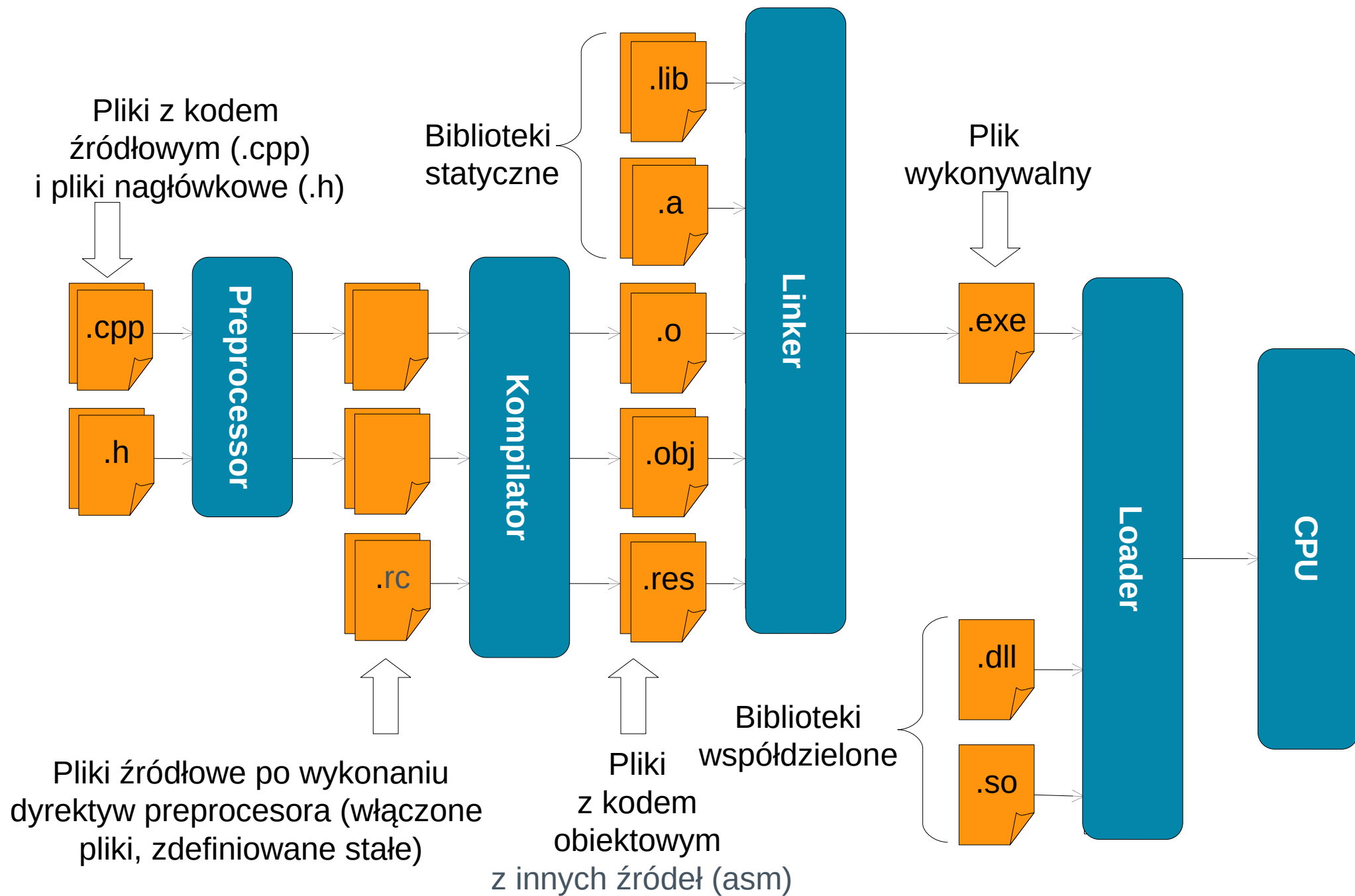


# Od kodu źródłowego do wykonania programu

- Kod programu należy przetłumaczyć do postaci zrozumiałej przez komputer, tzn. możliwej do wykonania przez procesor
- W przypadku C++ proces ten obejmuje kilka etapów:
  - Preprocessing kodu źródłowego, podczas którego preprocesor interpretuje dyrektywy (m.in. wstawia zawartość plików włączonych dyrektywą `#include`, usuwa fragmenty objęte klauzulą `#ifdef/#endif`, `#ifndef/#endif` i podstawia tekst zdefiniowany dyrektywami `#define`)
  - Właściwa kompilacja do postaci tzw. pliku obiektowego
  - Zasadniczo jeden plik źródłowy `.cpp` stanowi pojedynczą jednostkę kompilacji, kompilowaną do osobnego pliku obiektowego
  - Łączenie (linkowanie, konsolidacja) wszystkich plików obiektowych do pojedynczego pliku wykonywalnego

# Od kodu źródłowego do wykonania programu

- Zamiast pliku wykonywalnego wynikiem linkowania może być również:
  - Biblioteka statyczna
  - Biblioteka dynamiczna
- Biblioteka to plik zawierający wiele plików obiektowych, definiujących funkcjonalność, z której możemy skorzystać w swoich programach
  - Biblioteki statyczne (zwykle pliki z rozszerzeniem .a lub .lib) dołączamy do programu na etapie linkowania
  - Biblioteki dynamiczne (w systemie Windows pliki .dll, w systemie Unix .so) są dołączane do programu dopiero w momencie jego uruchomienia



# Od kodu źródłowego do wykonania programu

- Cały proces budowania wersji wykonywalnej programu możemy uruchomić z wiersza poleceń, np. za pomocą kompilatora g++ stanowiącego składnik GNU Compiler Collection

*g++ program.cpp -o program.exe*

Pierwszy argument, to nazwa pliku z kodem źródłowym

Po opcji -o (ang. output) podajemy nazwę pliku wynikowego

- w systemach z rodziny Unix rozszerzenie .exe nie jest konieczne. Programy wykonywalne muszą natomiast mieć ustawiony atrybut wykonalności. W powyższym przykładzie g++ ustawi go automatycznie, ale w razie potrzeby warto pamiętać, że służy do tego polecenie chmod

# Od kodu źródłowego do wykonania programu

- Poprzedni przykład, to bardzo prosty przypadek (cały program zawarty w pojedynczym pliku źródłowym) możliwe jest również podanie wielu plików źródłowych:

```
g++ pr1.cpp pr2.cpp pr3.cpp -o program.exe
```

UWAGA: Wszystkie etapy budowania pliku wykonywalnego (preprocessing, kompilacja, linkowanie) realizowane są tym jednym poleceniem

- Możemy jednak zobaczyć wyniki pośrednie używając odpowiednich opcji narzędzia g++:
  - E (wynik preprocessingu – w postaci tekstowej)
  - S (wynik kompilacji – podgląd kodu assemblera)
  - c (wynik kompilacji – w postaci plików obiektowych .o)

# Kompilatory dla języka C++

- › CLang (<http://clang.llvm.org/>)
- › Cygwin (<http://www.cygwin.com/>)
- › MinGW (<http://www.mingw.org/>)
- › GCC (<http://gcc.gnu.org/>)
- › Intel C++ (<https://software.intel.com/>)
- › Microsoft C++ (<https://visualstudio.microsoft.com>)
- › Watcom C++ (<https://sourceforge.net/projects/openwatcom/>)

# Od kodu źródłowego do wykonania programu

- Z punktu widzenia programisty, Python jest językiem interpretowanym, więc nie ma problemu z kompilacją kodu
- Programy (inaczej: skrypty) Pythona są wykonywane (interpretowane) instrukcja po instrukcji przez osobny program zwany interpreterem, stanowiący podstawowy składnik instalacji Pythona
- Wykonanie programu w Pythonie jest bardziej złożonym procesem, obejmującym również zwykle fazę kompilacji - szczegóły zależą od konkretnej implementacji Pythona:
  - CPython (implementacja referencyjna)
  - PyPy (kompilacja JIT, ang. Just-In-Time)
  - Jython (implementacja Pythona dla Java Virtual Machine, JVM)
  - IronPython (implementacja Pythona dla platformy .NET)



# Od kodu źródłowego do wykonania programu

Wykonanie skryptu Pythona przez referencyjny interpreter CPython przebiega w dwóch etapach, obejmujących:

- kompilację kodu źródłowego do postaci pośredniej, tzw. kodu bajtowego (ang. bytecode)
- wykonanie kodu bajtowego przez maszynę wirtualną (ang. Virtual Machine, VM)
- Kod bajtowy jest zwykle zapisywany w plikach z rozszerzeniem .pyc w bieżącym katalogu, ale może też być zapisany w folderze tymczasowym, lub bezpośrednio uruchomiony z pamięci operacyjnej (bez zapisywania na dysku)
- Jeśli podczas uruchomienia programu zostanie odnaleziony odpowiadający mu plik .pyc nowszy niż źródłowy plik .py (co oznacza, że kod nie był edytowany od ostatniego wykonania) – kompilacja jest pomijana i uruchamiana jest od razu poprzednio skompilowana wersja z pliku .pyc

# Od kodu źródłowego do wykonania programu

- Dzięki pośredniej formie uruchamiania skryptów Pythona, ich wykonanie może być znacząco szybsze niż bezpośrednia interpretacja kodu źródłowego (kod bajtowy jest zoptymalizowany i bardziej efektywny)
  - Wykonanie kodu bajtowego przez VM jest jednak zwykle wciąż istotnie wolniejsze niż wykonanie kodu maszynowego (np. skompilowanego programu C++) przez procesor
- Uwaga:
- pierwsze wykonanie programu w Pythonie może być znacząco wolniejsze, niż kolejne, z uwagi na konieczność kompilacji kodu do postaci pośredniej i wygenerowania pliku .pyc (który może być używany podczas kolejnych uruchomień)

# Wybrane środowiska programistyczne

- DevC++ (<https://sourceforge.net/projects/orwellddevcpp/>)
- C++ Builder Community (<https://www.embarcadero.com/>)
- Visual Studio Community (<https://visualstudio.microsoft.com>)
  
- PyCharm (<https://www.jetbrains.com/pycharm/>)
- Spyder (<https://www.spyder-ide.org/>)
  
- NetBeans (<https://netbeans.org/>)
- Eclipse (<https://www.eclipse.org/>)

# Środowiska online

- Colabulatory (<https://colab.research.google.com/>)
- REPL (<https://repl.it/languages/python3>)
- .NET Fiddle (<https://dotnetfiddle.net/>)
- C# online (<https://rextester.com/>)
- C++ shell (<http://cpp.sh/>)

# Czy warto uczyć się programowania?

Tak, ponieważ dzięki temu:

- poznamy zasady działania komputera, jego ograniczenia i możliwości,
- nauczymy się precyzyjnie formułować problemy do rozwiązania przy użyciu komputerów i nie tylko,
- nauczymy się rozwiązywać nietypowe zagadnienia,
- w pełni wykorzystamy dostępne oprogramowanie,
- nauczymy się pracy w zespole

# PYTANIA ?



<https://github.com/rroszczyk/Python>

[radoslaw.roszczyk@pw.edu.pl](mailto:radoslaw.roszczyk@pw.edu.pl)

Dziękuję