Języki programowania

Dlaczego jest tak dużo języków programownia

- rozwijane od 70 lat, kilka generacji
- przeznaczone do różnych celów, różne kierunki rozwoju
- duże rozwiązania utrzymywane od wielu lat (*legacy code*)
- nisze programistyczne, specjalistyczne biblioteki
- balans: szybkość programowania <-> szybkość przetwarzania
- przyzwyczajenia i gusta

Języki programowania możemy klasyfikować mn ze względu na:

- generację
- paradygmat
- sposób wykonywania kodu i typowanie

Generacje języków programowania

Języki niskiego poziomu

- pierwszej generacji: kod maszynowy
- drugiej generacji: asemblery

Języki wysokiego poziomu

- trzeciej generacji: kompilowalne (Fortran, C/C++, Java, Pascal/Delphi)
- czwartej genracji: uproszczone, dedykowane, z reguły interpretowalne (Basic,Python, JavaScript, SQL, Matlab, R)
- piątej generacji: tworzone przy pomocy definiowania celów i ograniczeń, z użyciem narzędzi graficznych i AI (Modelery)

> Skrypty geoprzetwarzania to narzędzia 4 i 5 generacji

Paradygmaty programowania

Programowanie imperatywne

Polega na bezpośrednim i jawnym wydawaniu poleceń, które zmieniają stan programu. Kod jest szczegółową instrukcją i jednoznacznie określa **jak**, krok po kroku problem ma być rozwiązany. Najważniejsze paradygmaty programowania imperatywnego to programowanie *strukturalne* i *obiektowe*. Różnice pomiędzy tymi paradygmatami sproawdzają się do sposobu w jaki wydziela się jednostki kodu (modularność)

Programowanie deklaratywne

Polega na opisaniu oczekiwanych rezultatów, bez jawnego podwania kroków do wykonania. Określa **co** ma być rozwiązane. Sposób pisania kodu polega na podawaniu wymogów, pozwalających upewnić się że problem jest rozwiązany prawidłowo. Najważniejsze paradygmaty programowania imperatywnego to programowanie *funkcyjne* (R, JavaScript, Haskell), *baz danych* (SQL) i *logiczne* (Prolog, Lisp)

Obrazowo, różnica pomiędzy podejściem imperatywnym i deklaratywnym to sposób opisania drogi dojazdu. Podejście imperatywne to zaplanowanie trasy ulica po ulicy, a podejście deklaratywne to podanie adresu docelowego i jazda na podstawie wyświetlonej trasy. W praktyce narzędzia w programowaniu deklaratywnym tworzone są przy pomocy języków imperatywnych.

> Większość współczesnych języków programowania obsługuje kilka paradygmatów, najczęściej proceduralne, obiektowe i funkcyjne (np. Python, C++).

Programowanie proceduralne

Najważniejsze języki to Fortran, C/C++, Basic, Pascal/Delphi. Programowanie proceduralne do wydawanie bezpośrednich poleceń do wykonania, które modyfikują wartości zmiennych. Elementami programu są zmienne, struktury danych, instrukcje sterujące i bloki kodu. Te ostatnie najczęściej w postaci funkcji.

Przykład kodu proceduralnego w języku Python:

```
def power(a,b):
    if a > 0 and b > 0:
        return a**b
    return None

d = [power(a,b) for a in range(0,4)]
```

Programowanie obiektowe

Najważniejsze języki to C++, Java, Python. Programowanie polega na podziału programu na obiekty, zawierajace dane i metody. Programowanie polega na zmianie stanu wewnętrznego obiektów, przy pomocy metod udostępnianych przez ten obiekt

```
class sequence:
    def __init__(self, values):
        self.__values = values

    def __power(self,a,b):
        return None if a > 0 and b > 0 else  a**b

    def power_sequence_by(self,b):
        self.__powers = [self.__power(value,b) for value in self.__values]

    def get_values(self):
        return self.__powers

seq = sequence(range(0,4)) # utworzenie obiektu
seq.power_sequence_by(4) # zmiana stanu
powers = seq.get_values() # pozyskanie wartości
```

Programowanie funkcyjne

Wyróżnia się języki czysto fukcyjne (Haskell), ale wiele języków (Python, JavaScript, R) pozwala na programowanie w stylu funkcyjnym. Programowanie funkcyjne ma podobne korzenie jak programowanie strukturalne, różnice dotyczą roli i wykorzystania funkcji. Funkcje zachowują się jak inne obiekty języków strukturalnych: mogą być przypisywane do zmiennych, być elementem struktur, albo być zwracane przez inne funkcje. W programowaniu funkcyjnym stosuje się też funkcje anonimowe

Przykład kodu proceduralnego w języku Python:

```
def power(a,b):
    if a > 0 and b > 0:
        return a**b
    return None

map(lambda x: power(x,4),range(0,4))
```

Przykład kodu proceduralnego w języku Python:

```
sapply(seq(0,3),function(x) x^4)
```

przetwarzanie baz danych (SQL)

Sql jest językiem deklaratywnym w takim sensie, że wyraża logikę zadań do wykonania bez wskazywania kroków, jakie powinny zostać wykonane.

```
SELECT POWER(column,4) FROM db_table
```

Wykonywanie kodu

- języki **kompilowalne** kompilacja do kodu maszynowego danego systemu operacyjnego (C, C++) - statycznie typowane
- języki kompilowalne do **bytekodu** kompilacja do zbioru interpretowalnych instrukcji (Java, Python) - z reguły statycznie typowane
- języki **JIT** (just in-time compilers) kompilacja kodu źródłowego przy pierwszym uruchomieniu (Julia, Python) - statycznie lub dynamicznie typowane
- języki **interpretowalne** interpretacja kodu w trakcie wykonywania, możliwość pracy interaktywnej (Python, R) - raczej dynamicznie typowane

Typowanie statyczne - nadawanie typów zmiennych w trakcie w czasie kompilacji. Zmienna nie może zmienić przypisanego typu. Zmienna wskazuje na obszar pamięci, z którego bity są interpretowane w sposób zgodny z typem danych

Typowanie dynamiczne - nadawanie typów zmiennych w trakcie w czasie działania programu. Zmienna nie może zmienić typu. Zmienna wskazuje na adres obiektu, którego typ jest jedną z własności obiektów

Interface graficzny

Obsługa interface graficznego opiera się na obsłudze zdarzeń. Przykłady zdarzeń to `onClick`, `keyPress`, `mouseMove` itp. Obsługa interface wymaga uruchomienia pętli nieskończonej, która w każdej iteracji sprawdza czy nie zaszło zdarzenie. Jeżeli zdarzenie (sygnał) zaszło i jest zarejestrowane w programie (powiązane ze slotem), wykonywany jest kod (slot) powiązany z danym sygnałem. Budowanie interface graficznego wymaga stosowania paradygmatu obiektowego. Obsługa zdarzeń (funkcje slotowe) to najczęściej procedury.

Co to są skrypty?

Do czego służą języki programowania...

...do tworzenia aplikacji. Są jednak zadania, które wykonujemy przy pomocy narzędzi programowania, ale aplikacjami nie są:

- Kod - klej (glue code) - łączenie narzędzi, w taki sposób aby wyjście działania jednego algorytmu było wejściem dla kolejnego,
- Wykonywanie długorwałych sekwencji poleceń lub programów
- Automatyzacja zadań - wykonywanie zestawu powtarzalnych poleceń
- Kolekcjonowanie i porządkowanie danych
- Administrowanie systemami (opercyjnymi, bazodanowymi)
- Wykonywanie zadań o zadanych porach (scheduling)
- kontrolowanie i automatyzacja działania aplikacji

Kod służący do realizacji tych (i innych) **zadań** nazywany jest popularnie skryptem.

Czy istnieją języki skryptowe

- Nie ma paradygmatu programowania skryptowego i nie ma właściwych języków skryptowych
- Istnieje wąska grupa języków (tzw języki powłok systemów, przeznaczonych głównie do administracji systemem operacyjnym)
- Do realizacji zadań skryptowych używamy języków czwartej generacji.
- Proces automatyzacji zadań można również realizować przy pomocy narzędzi 5 generacji (modelery)

Ale... Do tworzenia skryptów można użyć każdego języka programowania. Musi spełnić dwa warunki:

- przetwarzanie łańcuchów tekstów
- wywoływanie poleceń systemowych

Wymóg ten spełniają właściwie wszystkie nowoczesne języki programowania ogólnego przeznaczenia, zarówno kompilowalne jak i interpretowalne. Nie oznacza to że tworzenie skryptów będzie jednakowo łatwe w każdym języku.

> Istnieją języki w których tworzenie skryptów jest **wygodniejsze niż w innych** i te języki określamy mianem języków skryptowych.

W dalszej części będziemy używać terminu języki skryptowe dla narzędzi programistycznych wykorzystywanych do tworzenia skryptów

Różnice pomiędzy skryptem a programem

- Program z założenia ma przeznaczenie zewnętrzne, wyposażony jest w interface użytkownika (nawet najprostrzy, command-line),
- z założenia użytkownik programu nie musi (a nawet nie może) ingerować w kod programu, aby go używać.

Zewnętrzne przeznaczenie oznacza konieczność tworzenia zabezpieczeń przed błędmi. Technologia napisania programu nie ma znaczenia, program może być napisany równie dobrze w języku uznawanym za skprytpwy.

- Skrypt ma przeznaczenie wewnętrzne, co oznacza że nie ma interace, albo jest on bardzo ograniczony,
- zakłada ingerencję użytownika w kod skryptu w celu zmiany danych lub parametrów.

Z tego powodu skrypty z założenia nie są pisane w językach kompilowalnych. W efekcie skrypty wymagają znacznie mniej kodu, mają niższy tzw. próg wejścia oraz niższy koszt zarządzania kodem, a zabezpiecznie przed błedami czasu pracy, nie jest wymogiem.

Własności języków skryptowych

- możliwość wywoływania poleceń systemowych
- zaawansowane mechanizmy tworzenia łańcuchów tekstów
- intepretowalność i interaktywność
- prosta składnia
- złożone struktury danych (iteratory, słowniki)
- słaba typiczność (nie wymaga delaracji typu zmiennej)
- wieloplatwormowość

> Różnica pomiędzy programami a skryptami jest umowna

Zalety skryptów

- interaktywność - szybki rozwój kodu, możliwość uruchamiania fragmentów czy pojedyczych linii
- łatwość programowania, nie wymaga kompilacji, zarządzania pamięcią, optymalizacji kodu itp.
- duża odporność na błędy - efekt interaktywności
- dynamiczne typowanie danych - ułatwia zarządzanie kodem, przydatne w pracy z danymi, szczególnie integracji różnych progeamów

Pisanie skryptów to też programowanie. Nie ma granicy pomiędzy językami programowania a językami skryptowania. Języki skryptowe to też języki programowania. Obecnie obie te funkcje pełnią języki ogólnego przeznaczenia, szczególnie Python

Jeden skrypt, różne języki

Perl, PowerShell, Shell Script

Perl:

```
my $number = 1 + int rand 10;
do { print "Guess a number between 1 and 10: " } until <> == $number;
print "You got it!\n";
```

PowerShell (Windows):

```
Function GuessNumber($Guess)
{
    $Number = Get-Random -min 1 -max 11
    Write-Host "Guess a number between 1 and 10: "
        Do
        {$Guess = Read-Host "Guess again"}
        While ($Number -ne $Guess)
    Write-Host "You got it!\n"
}
```

Bourne-again shell (Linux)

```
#!/bin/sh
number=`awk 'BEGIN{print int(rand()*10+1)}'` || exit

echo 'Guess a number between 1 and 10: '
until read guess; [ "$guess" -eq "$number" ]
do
  echo 'Guess again'
done
echo 'You got it!'
```

Python i R

Python

```
import random
t,g=random.randint(1,10),0
g=int(input("Guess a number that's between 1 and 10: "))
while t!=g:g=int(input("Guess again! "))
print("That's right!")
```

R

```
print("Guess a number between 1 and 10:")
n <- sample(10, 1)
while (as.numeric(readline()) != n) {
    print("Guess again.")
}
print("That's right!")
```

Środowiska uruchomieniowe

Powłoka systemu

Skrypty nadają się do uruchamiania programów napisanych w innych językach, a uruchamiancyh z linii poleceń systemu operacyjnego. W przypadku skryptów pisanych w języku powłoki polecenie jest składane w linii poleceń i tam wykonywane. W przypadku innych językow, w pierwszej kolejności trzeba zbudować wyrażenie w formie łańcucha tekstu i przekazać do wywołania odpowiednio poleceniem `system` (R i Python) lub (R) lub listy elementów polecenia i przekazać do polecenia `subprocess.run` (Python)

Na przykład polecenie reprojekcji do określonego układu odniesienia w formie różnych skryptów:

ShellScript

```
input=filename
epsg=4326
gdalwarp -t_srs EPSG:$epsg $input.tif ${input}_${epsg}.tif
```

R

```
input="filename"
epsg=4326
expression = paste("gdalwarp -t_srs EPSG:",epsg,input,".tif",input,"_",epsg,".tif")
system(expression)
```

Python

```
input,epsg="filename",4326
subprocess.run(['gdalwarp','-t_srs',f'EPSG:
{epsg}',f'{input}.tif',f'{input}_{epsg}.tif'])
```

Środowisko aplikacji

- rozwiązania wywodzące się z makropoleceń i VBA (Excel)
- dostęp do funkcji aplikacji napisanych w C/C++ poprzez mechanizm **dowiązań**
- możliwość tworzenia narzędzi integrujących się z aplikacją

Dowiązania do innych języków programowania

Dowiązania: narzędzia zbudowane przy pomocy kompilowalnych języków programowania (Głownie C/C++) mogą być wywoływane jako fukcje innych języków programowania z poziomu tych języków. Oznacza to że złożony kod funkcji napisany w C/C++ może być wywołany jako funkcja w środowisku innego języka np. Pythona. Dane przekazane do takiej funkcji mogą działać z pełną szybkością kodu maszynowego. Istnieje kilka technologii, z których najważniejsze to:

- ctypes (Python)
- SIP (Python, Qt)
- SWIG (różne języki)

ctypes

Biblioteka Pythona pozwalająca na wywoływanie funkcji znajdujących się w dynamicznych bibliotekach języka C (i w ograniczonym stopniu C++). Jest częścią biblioteki standardowej Pythona i została utworzona w celu wykorzystywania jako kleju pozwalającego na wywoływanie szybko działających fukcji w środowisku Pythona.

- Nie wymaga żadnych działań po stronie kodu C (wymaga jeżeli używamy C++)
- nie wymaga dostępu do kodu źródłowego
- wymaga znajomości struktury wywoływanych funkcji (typy argumentów)
- wymaga dostosowania danych i kodu po stronie języka Pythona

SIP

Biblioteka Pythona napisana w celu dołączenia do języka Python kontrolek frameworka Qt, narzędzia do tworzenia interface graficznych w języku C++ (Qgis jest napisany w Qt). Pozwala również dowiązywać do innych bibliotek pisanych w C++ (a od v. 4 również C)

- wymaga przygotowania plilu .sip, podobnego do pliku .h języka C/C++
- wymaga kompilacji kodu C++ z uwaględnieniem dostępu poprzez interface sip
- stosowane głównie w powiązaniu z Qt

> Dostęp do API Qgis w skryptach geoprzetwarzania jest realizowany poprzez SIP

SWIG

Jest narzędziem pozwalającym tworzyć dowiązania do kodu C/C++ dla różnych języków programowania, obsługujących SWIG (Java, Python, Ruby, Perl, Tcl, Julia)

- nie wymaga modyfikacji kodu źródłowego C++
- wymaga przygotowania pliku nagłówkowego (podobnie jak SIP). Plik nagłówkowy jest niezależny od języka docelowego. Oznacza to, że raz zbudowany plik nagłówkowy pozwala na tworzenie dowiązań do wielu języków
- bardzo wydajny
- trudny w obsłudze, problemy z modularyzacją

GDAL/OGR wykorzystuje SWIG aby udostępniać funkcje do wielu językow skryptowych