Algorytmy

Algorytm - zbiór precyzyjnych instrukcji wraz z regułami, w jakiej kolejności mają być wykonywane, oraz kiedy ich wykonywanie należy zakończyć.

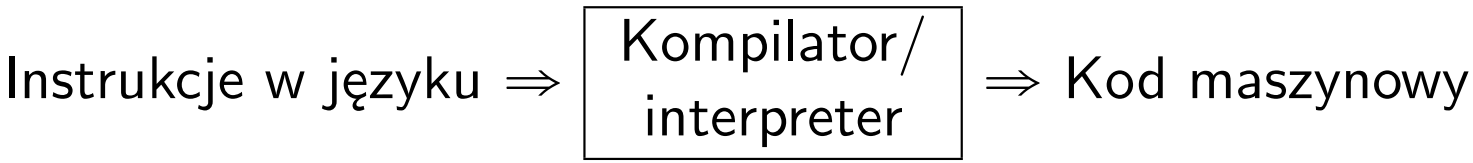
„Precyzyjna instrukcja" może nie mieć sensu na danej platformie - procesory są w stanie wykonywać jedynie proste operacje arytmetyczne, podczas gdy w algorytmach często stosujemy bardziej złożone obiekty (np. macierze, napisy, obrazy, ...).

Język programowania - notacja pisania programów, czyli opisów algorytmów.

Idea: programy mogą dopuszczać instrukcje, które nie mają odpowiedników w kodzie maszynowym; definiowanie złożonych obiektów, oraz operacje na nich.

Języki programowania

Programy napisane w danym języku muszą zostać przetłumaczone na język maszynowy, aby mogły zostać wykonane. Służy do tego oprogramowanie: **kompilatory** i **interpretery**.



Tłumaczenie może odbywać się w krokach, używając języków pośrednich (np. C# ⇒ CIL ⇒ interpretacja i produkcja kodu maszynowego just-in-time).

Fundamentalne różnice między interpreterem a kompilatorem - materiał na inny wykład.

Języki programowania

Języki programowania (języki naturalne też) - opis przez składnię (syntax) i semantykę (semantics).

Składnia - reguły stanowiące, co jest poprawnym (albo: „legalnym") wyrażeniem lub instrukcją w języku.
Przykład: wyrażenia arytmetyczne składają się z liczb, zmiennych, symboli operatorów arytmetycznych, nawiasów. Istnieją reguły konstrukcji poprawnych wyrażeń (operatory muszą pojawiać się między podwyrażeniami, nawiasy otwarte muszą zostać zamknięte, etc.).

Semantyka - znaczenie poprawnych wyrażeń języka.
„$a + b$" to wyrażenie, którego wartość jest liczbą będącą sumą a i b.

Obiekty i typy

W większości języków programowania pracujemy z **obiektami** (liczby, napisy, obrazy, listy zakupów, gatunki książek, ...).

W Pythonie, każdy obiekt ma **typ**. Typ obiektu determinuje jakie wartości może przyjmować obiekt, i jakie operacje można na nim wykonać.

Niektóre elementarne, wbudowane typy w Pythonie:

Nazwa	**Znaczenie**	**Przykładowe wartości**
int	liczba całkowita	1337, -1
float	liczba zmiennoprzecinkowa	2.5, -7e100
bool	wartość logiczna	True, False (jedyne)
str	napis (string)	"abc123", "X Y Z"
NoneType	„brak wartości"	None (jedyna)

Wyrażenia arytmetyczne

Wyrażenia arytmetyczne - wyrażenia reprezentujące liczby.

Skonstruowane z:

- operatorów arytmetycznych, liczb, nawiasów,
- nazw reprezentujących obiekty (o tym zaraz),
- pewnych podwyrażeń (o tym później).

Operatory arytmetyczne (w kolejności wykonywania):

1. ** (potęgowanie),
2. -, + (jednoargumentowy minus i plus),
3. *, /, %, // (mnożenie, dzielenie, modulo, dzielenie całkowite),
4. +, -,
5. ...

Przypisanie

Nazwa - identyfikator obiektu. Dowolnemu obiektowi można nadać nazwę i używać jej w kodzie, reprezentując ten obiekt.

Podstawowa składnia:

```
nazwa = wyrazenie
```

np.

```
r = 5
area = 3.14 * r ** 2
```

Wartość wyrażenia z prawej strony wyliczana jest raz - w momencie przypisania.

Wyrażenia a instrukcje

Dwa rodzaje „napisów”: instrukcje (statements) i wyrażenia (expressions).

Instrukcja - powoduje zmianę stanu programu, nie posiada wartości (np. przypisanie)

Wyrażenie - wylicza się do pewnej wartości (tzn. obiektu pewnego typu).

Wyrażenia ogólnie

Wyrażenia konstruowane z:

- operatorów, stałych, nawiasów,
- nazw reprezentujących obiekty,
- podwyrażeń.

Niektóre operatory (w kolejności wykonywania):

1. arytmetyczne,
2. $<=$, $<$, $>$, $>=$ - nierówności,
3. $==$, $!=$ - równość, nierówność,
4. not - negacja,
5. and - koniunkcja,
6. or - alternatywa.

Wejście/wyjście

Wejście/wyjście: wbudowane w język input() i print().

```
print(wyrazenie1, wyrazenie2, ...)
```

np.

```
print("Hello", "World!", 12 + 10)
>> Hello World! 22
```

```
s = input("Podaj napis")
```

Konwersja typów:

```
n = int(2.5)
```

Np. str → int, str → float:

```
n = int(input("Podaj liczbe calkowita"))
x = float(input("Podaj liczbe rzeczywista"))
```

Przepływ sterowania

Pewne sposoby kontroli kolejności wykonywania instrukcji:
Instrukcja warunkowa.

```
if wyrazenie: instrukcja
```

```
if wyrazenie:
        instrukcja1
        instrukcja2
        ...
```

```
if wyrazenie1:
        instrukcja1
        instrukcja2
        ...
elif wyrazenie2: # dowolna ilosc razy
        instrukcja3
        ...
else:
        instrukcja4
        ...
```

Przepływ sterowania

Pewne sposoby kontroli kolejności wykonywania instrukcji: pętle* (pierwsza przymiarka)

```
while wyrazenie: instrukcja
```

```
while wyrazenie:
        instrukcja1
        instrukcja2
        ...
```

```
for nazwa in range(n, m): instrukcja
```

```
for n in range(1, 10):
        print(n)
```

* - pętla while będzie na drugim wykładzie.