Informacje o wykładzie

- *Programowanie 1 (Python)*
- Nie zakłada wstępnej znajomości Pythona ani programowania.
- Strona wykładu:

  math.uni.wroc.pl/~jagiella/p1python/

  (lista omówionych zagadnień, materiały z wykładu, przydatne linki)
- Strona ćwiczeń (laboratoriów):

  math.uni.wroc.pl/~jagiella/p1python/lab/

  (listy zadań, przykładowe rozwiązania)
- Konsultacje: TBA.

Strony formalne i zaliczenie

- Wykład kończy się praktycznym (pisemnym) egzaminem.
- Dopuszczenie do egzaminu: zaliczenie laboratoriów przez wykonywanie list zadań.
- Listy zadań mają charakter zadania domowego i należy je wykonywać samodzielnie.
- Listy zadań będą pojawiać się po każdym wykładzie i obowiązują (domyślnie) na następny tydzień.
- Prowadzący mogą modyfikować harmonogram w swojej grupie lub stosować własne listy.
- 50% na 3.0, 60% na 3.5, ..., 90% na 5.0.

Strony techniczne, ćwiczenia

Pierwsze ćwiczenia:

- Instalacja Pythona (i ewentualnych innych narzędzi).
- Instrukcja przekazywania rozwiązanych zadań.
- Sposoby rozliczania i oceniania rozwiązań.
- Ewentualne inne kanały informacyjne (zespoły na MS Teams, ...).

Pierwsza lista pojawi się po wykładzie i obowiązuje na 10-16.10.2023 (wtorek-poniedziałek).

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**.

$$\text{Instrukcje w języku} \Rightarrow \boxed{\begin{array}{c}\text{Kompilator/}\\ \text{interpreter}\end{array}} \Rightarrow \text{Kod maszynowy}$$

Tłumaczenie może odbywać się w krokach, używając języków pośrednich, np.:

- C# $\Rightarrow$ CIL $\Rightarrow$ interpretacja i produkcja kodu maszynowego just-in-time).
- Python $\Rightarrow$ pośredni kod binarny $\Rightarrow$ interpretacja/interpretacja just-in-time.

Fundamentalne różnice między interpreterem a kompilatorem – materiał na inny wykład (lub kurs).

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.
Przykład: „$a + b$" to wyrażenie, którego wartość jest liczbą będącą sumą a i b.

Obiekty i typy

W większości języków programowania pracujemy z **obiektami** (liczby, napisy, macierze, obrazy, dźwięki, listy innych obiektów, ..., lub bardziej konkretnie: widoki baz danych, dane bibliograficzne książek, reprezentacje „prawdziwych" obiektów w danej symulacji, ...

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	0, 1, 2, 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 plus i minus),
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 = wyrażenie
```

np.

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

Wartość wyrażenia z prawej strony wyliczana jest raz – w momencie przypisania. Sama nazwa może zostać później przypisana do innego obiektu.

Podobny model: Java, C#, R (nazwa to etykieta).
Inny model: C, C++ (nazwa/zmienna to sam obiekt).

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,
- ...

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 liczbę calkowitą"))
x = float(input("Podaj liczbę rzeczywistą"))
```

Przepływ sterowania

Pewne sposoby kontroli kolejności wykonywania instrukcji:
Instrukcja warunkowa.

```
if wyrażenie: instrukcja
```

```
if wyrażenie:
    instrukcja1
    instrukcja2
    ...
```

```
if wyrażenie1:
    instrukcja1
    instrukcja2
    ...
elif wyrażenie2: # dowolna ilość razy
    instrukcja3
    ...
else:
    instrukcja4
    ...
```

Przepływ sterowania

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

```
while wyrażenie: instrukcja
```

```
while wyrażenie:
    instrukcja1
    instrukcja2
    ...
```

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

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

* – pętla while pojawi się na drugim wykładzie.