PODSTAWY PROGRAMOWANIA W JĘZYKU PYTHON

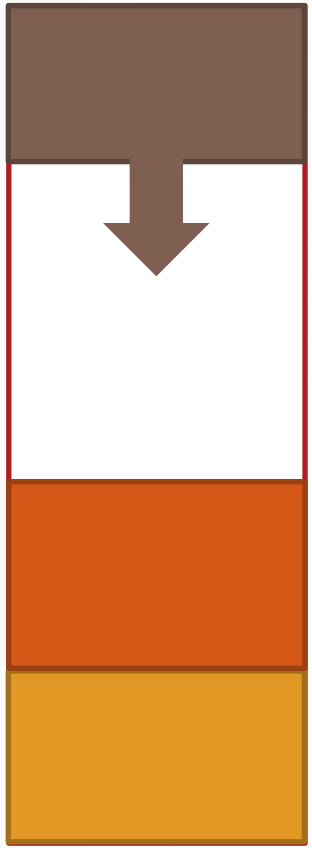
Dzień 2

Agenda

- Pamięć w komputerze
- Typy danych, zmienne
- Operatory
- Funkcje pomocnicze
- Instrukcje warunkowe
- Code style

Pamięć komputera

Pamięć w programach



Pamięć w Python

Zmienne i operatory

Podstawowe typy danych

- **`None`** – nic, null, brak
- 123 - **`int`** - liczby całkowite (integer)
- 54.45 - **`float`** - liczby zmiennoprzecinkowe (floating point)
- "Ala #23" - **`str`** - łańcuchy znaków (string)
- True/False - **`bool`** - prawda/fałsz (boolean)

Zmienna

- Nazwany obszar pamięci, w którym znajduje się jakaś wartość.
- Pozwala na ponowne użycie wartości w innym miejscu w kodzie.
- Zbiór dostępnych zmiennych nazywamy **przestrzenią nazw**.
- Przypisanie realizowane jest przez pojedynczy znak równości “=”.

```
>>> my_value = 124
>>> nazwisko = "Kowalski"
>>> czy_obecny = True
```

Operatory

Operatory

Matematyczne:

```
+, -, *, /, //, %, **
```

Logiczne:

```
==, !=, <, >, <=, >=, in, is, and, or, not
```

Przypisanie, kolejność wykonywania działań

=

- Python wykonuje wyrażenia od lewej do prawej.
- W przypadku przypisywania,
  najpierw wykonywane są operacje z prawej strony znaku przypisania (obliczenie wartości),
  później z lewej strony.
- Kolejność wykonywania można sprawdzić tutaj:
  https://docs.python.org/3/reference/expressions.html#operator-precedence

```
>>> wynik = 5 != 4 and 'a' not in 'Andrzej'
```

Operator porównania

= VS ==

- **=** przypisuje wartość do zmiennej

```
>>> x = 1
```

- **==** porównuje dwie wartości (zwraca True lub False)

```
>>> 1 == (2 - 1)  # True
```

Komentarze

#

Wszystko po tym znaku jest ignorowane przez interpreter.
Może służyć do opisania fragmentu kodu lub jego “*wykomentowania*” (tymczasowego usunięcia).

Funkcje pomocnicze

Funkcja

- Blok wydzielonego kodu, używany do wykonania określonego działania.
- Funkcje zapewniają modułowość kodu oraz ułatwiają jego ponowne użycie.
- Python posiada funkcje wbudowane oraz pozwala na ich definiowanie.
- Wywołanie funkcji:

```
nazwa_funkcji()
nazwa_funkcji(argumenty)
```

```
>>> type("Ala ma kota")
```

Funkcje input i print

```
>>> nazwisko = input(”Podaj nazwisko: ”)
```

input(prompt_text) przyjmuje od użytkownika dane i zapisuje je do zmiennej o typie string.

```
>>> print(nazwisko)
```

print() służy do wypisania tekstu na ekranie.
Automatycznie dodaje na końcu stringa znak specjalny nowej linii *\n*.

Rzutowanie, czyli konwersja typu

- Każda zmienna posiada **typ**, który określa rodzaj informacji w niej przechowywanej oraz operacje jakie można na niej wykonać.
- Czasami potrzebna jest zmiana typu zmiennej (rzutowanie, *cast)* w celu wykonania na niej operacji charakterystycznej dla typu np. zamiana *wejścia* użytkownika ze *stringu* na *inta* w celu wykonania operacji arytmetycznej.

```
nazwa_typu(wartosc)
```

```
>>> int("4")  # 4
>>> str(4)  # "4"
```

Rzutowanie c.d.

- Rzutowanie może spowodować utratę informacji (rzutowanie *floata* na *inta*) lub nawet błąd (rzutowanie litery na liczbę całkowitą).

```
>>> int(2.5) # 2
>>> int("Ala")  # ValueError
```

Metody wbudowane typów

Każdy typ danych (*string*, *integer*) posiada zdefiniowane metody (funkcje), które pozwalają na wykonanie różnych (najpopularniejszych) działań, właściwych dla tego typu.

```
typ.funkcja()
>>> "ala ma kota".capitalize()
```

Dokumentacja offline

- **`help()`** – wyświetla dokumentację / pomoc modułu, funkcji
- **`dir()`** - wyświetla składniki modułu

Instrukcje warunkowe

Blok kodu

Dwukropek rozpoczynający blok

```
Instrukcja/wyrazenie:
    Instrukcja
    Instrukcja
    Instrukcja/wyrazenie:
        Instrukcja
    Instrukcja
    I tak dalej...
```

Poziom 1
(4 spacje)

Poziom 2.
(8 spacji)

Blok kodu

- W Pythonie fragmenty kodu wydzielane są przez wcięcie tekstu (indentacje), nie klamry/znaki specjalne.
- Instrukcje poprzedzone takim samym wcięciem z białych znaków (tabulacji lub spacji) zawierają się w jednym bloku (dotyczą jednego zbioru operacji np. w pętli).
- Ważne jest aby wcięcie było zawsze konsekwentnie stosowane (rodzaj/liczba znaków).
- PEP 8 (Python Enhancement Proposal, *https://www.python.org/dev/peps/pep-0008/*) sugeruje używanie 4 spacji jako pojedynczego wcięcia.

Instrukcje warkunkowe

1. Weź książkę telefoniczną.
2. Otwórz książkę na środku.
3. Sprawdź nazwiska na otwartych stronach.
4. **Jeśli ”Wojtkowiak” jest wśród osób:**
    a. Zadzwoń do niego
5. **W przeciwnym razie jeśli ” Wojtkowiak” jest wcześniej w książce:**
    a. Otwórz lewą połowę po środku.
    b. Idź do kroku 3.
6. **W przeciwnym razie jeśli ” Wojtkowiak” jest później w książce:**
    a. Otwórz prawą połowę po środku.
    b. idź do kroku 3.
7. **W przeciwnym razie:**
8. poddaj się.

Instrukcja warunkowa

```
if warunek:
    # kod wykonany gdy warunek prawdziwy
elif inny warunek:
    # kod wykonany gdy warunek w if był fałszywy
    # warunek w tym elif musi być prawdziwy aby ten kod wykonać
elif (inny warunek1 and inny warunek2):
    # elif-ów może być wiele lub nie być żadnego,
    # kod wewnątrz elif wykona się tylko gdy wszystkie
    # poprzedzające go warunki były niespełnione (fałszywe)
else:
    # przypadek domyślny, nie ma warunku,
    # kod w else będzie wykonany gdy wszystkie poprzednie warunki
    # byly fałszywe, else moze byc tylko jeden lub wcale
```

Tablica logiczna

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Code Style

Zen of Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.

PEP
(Python Enhancement Proposal)

PEP 8 (stylowanie):https://www.python.org/dev/peps/pep-0008/

Podsumowanie

- Pamięć w komputerze
- Typy danych, zmienne
- Operatory
- Funkcje pomocnicze
- Instrukcje warunkowe
- Code style

Thanks!!