

# Lekcja

## Temat: Wstęp do programowania w Python

### Historia języka programowania Python

Python został stworzony przez **Guido van Rossuma** w latach 1989-1991 w Holandii. Nazwa nie pochodzi od węża, lecz od brytyjskiej grupy komediowej **Monty Python**, której Guido był fanem.

Prace nad nim rozpoczęły się pod koniec **1989** roku w **Centrum Matematyki i Informatyki** (CWI) w Amsterdamie, gdzie van Rossum szukał projektu, który mógłby zająć go w okresie świątecznym. **Python miał być następcą języka ABC**, który również rozwijano w CWI, ale van Rossum chciał stworzyć coś bardziej uniwersalnego i przyjaznego dla programistów.

### Kluczowe etapy rozwoju:

- **1991: Premiera pierwszej wersji** – 20 lutego 1991 roku ukazała się publiczna wersja 0.9.0. Python był od początku projektem open source, co pozwoliło na szybki rozwój dzięki wkładowi społeczności.
- **Lata 90.: Rozwój w CWI i CNRI** – Do 1995 roku van Rossum rozwijał Pythona w CWI, a następnie przeniósł się do Corporation for National Research Initiatives (CNRI) w USA, gdzie wydał wersje do 1.6 włącznie. W tym okresie język zyskał kompatybilność z licencją GPL.
- **2000: Python 2.0 i BeOpen.com** – Van Rossum i zespół przenieśli się do BeOpen.com, gdzie wydano Pythona 2.0. Wkrótce potem powstała Python Software Foundation (PSF), która do dziś zarządza rozwojem języka.
- **2008: Python 3.0** – Wprowadzono znaczące zmiany, takie jak lepsza obsługa Unicode, co jednak spowodowało niekompatybilność z Pythonem 2.x. Przejście na "trójkę" trwało lata, a wsparcie dla Pythona 2 zakończyło się w 2020 roku.
- **2018: Rezygnacja van Rossuma** – Guido van Rossum, znany jako "Benevolent Dictator for Life" (BDFL), zrezygnował z roli lidera. Od tego czasu rozwój nadzoruje Steering Council wybrany przez społeczność.

### Instalacja interpretera

#### Windows

1. Wejdź na <https://www.python.org/downloads/windows/>
2. Pobierz **Windows installer (64-bit)**
3. Zainstaluj (opcje standardowe)
4. Wykonaj polecenie: **py -3 -version** . Jeśli zainstalowałeś, poprawnie powinieneś otrzymać komunikat: **Python 3.14.2**
5. Kliknij **Win + I**
6. Kliknij **Aplikacje**, wybierz **Zaawansowane ustawienia aplikacji**. Po czym kliknij **Alias wykonywania aplikacji**
7. Zobaczysz listę. **Przewiń w dół i znajdź:**
  - python.exe
  - python3.exe

Przy nich są przełączniki (ON / OFF). Wyłącz dwie opcje python.exe, python3.exe

## Uruchomienie skryptu

Wykonaj polecenie, w katalogu skryptu Python:

***python nazwa\_skryptu.py***

## Rozszerzony przykład Hello World

```
import sys
print("Hello, World!")

print("Witaj w świecie Pythona!")
print(f"Używasz Pythona w wersji: {sys.version}")

imie = input("Jak masz na imię? ")
print(f"Cześć {imie}! Miło Cię poznać!")
```

## Podstawowe typy komentarzy

### Komentarz jednowierszowy - zaczyna się od #

```
# To jest komentarz jednowierszowy
x = 10 # To też jest komentarz - po kodzie
```

### Komentarz wielowierszowy - używamy trzech cudzysłowów ''' lub """

```
"""
To jest komentarz wielowierszowy.
Może zawierać wiele linii tekstu.
Często używany na początku pliku
lub do dokumentowania funkcji.
"""

'''
To też jest poprawny
komentarz wielowierszowy.
Używaj zgodnie z konwencją projektu.
'''

def oblicz_srednia(liczby):
    """
    Funkcja oblicza średnią arytmetyczną z listy liczb.

    Args:
        liczby (list): Lista liczb całkowitych lub zmiennoprzecinkowych

    Returns:
        float: Średnia arytmetyczna
    """
    return sum(liczby) / len(liczby) if liczby else 0
```

## Oznaczanie TODO i FIXME

*# TODO: Dodać obsługę błędów dla pustej listy*  
*# FIXME: Naprawić wyciek pamięci przy dużych danych*  
*# HACK: Tymczasowe rozwiązanie - wymaga refaktoryzacji*  
*# NOTE: Ważna uwaga dotycząca wydajności*  
*# OPTIMIZE: Można przyspieszyć przez caching*

```
def przetworz_dane(dane):  
    # TODO: Implementować walidację danych wejściowych  
    # FIXME: Dla pustych danych zwracamy None - do poprawy  
    if not dane:  
        return None # FIXME: Powinien być wyjątek  
  
    # ... reszta kodu ...
```

## Wcięcia Pythonie:

```
if x > 0:  
    print(x)
```

- **Dwukropek** : mówi: zaraz będzie blok
- **Wcięcie** mówi, co należy do tego bloku

### Przykład błędu

```
x = 5  
if x > 0:  
print("x jest dodatnie") # Błąd: brak wcięcia
```

Wynik:

**IndentationError: expected an indented block**

### Poprawnie

```
x = 5  
if x > 0:  
    print("x jest dodatnie") # wcięcie = 4 spacje (standard)
```

## 1. Typy numeryczne

### int - liczby całkowite

```
# Przykłady  
liczba_calkowita = 42  
ujemna = -10  
duza_liczba = 1_000_000 # podkreślniki dla czytelności, równoważne 1000000  
binarna = 0b1010 # 10 w systemie dziesiętnym  
""" od prawej liczymy  
    •  $1 \times 2^3 = 8$   
    •  $0 \times 2^2 = 0$   
    •  $1 \times 2^1 = 2$   
    •  $0 \times 2^0 = 0$ 
```

```
"""
```

```
szesnastkowa = 0xFF # 255
```

```
# Zastosowania: liczniki, ID, indeksy, obliczenia finansowe
```

```
cena_produktu = 299
```

```
ilosc_sztuk = 5
```

```
id_uzytkownika = 12345
```

## float - liczby zmiennoprzecinkowe

```
# Przykłady
```

```
pi = 3.14159
```

```
temperatura = -5.5
```

```
duza_liczba = 1.23e6 #  $1.23 \times 10^6 = 1230000.0$ 
```

```
# Zastosowania: obliczenia naukowe, finanse, pomiary
```

```
cena_netto = 19.99
```

```
waga = 2.5
```

```
srednia = 4.75
```

## complex - liczby zespolone

```
# Przykłady
```

```
liczba_zespolona = 3 + 4j
```

```
inna = complex(2, -3) # 2 - 3j
```

```
# Zastosowania: inżynieria, fizyka, przetwarzanie sygnałów
```

```
impedancja = 50 + 100j # zamiast i używa się j (zgodnie z konwencją inż. elektryków).
```

## 2. Typy tekstowe

### str - ciągi znaków

```
# Przykłady
```

```
imie = "Anna"
```

```
nazwisko = 'Kowalska'
```

```
wielolinijkowy = """To jest
```

```
wielolinijkowy
```

```
tekst"""
```

```
f_string = f"Witaj {imie}!" # f-string (Python 3.6+)
```

```
# Zastosowania: komunikaty, dane tekstowe, parsowanie
```

```
email = "user@example.com"
```

```
wiadomosc = "Dziękujemy za zakupy!"
```

```
sciezka = "C:/Users/Dokumenty"
```

## 3. Typy sekwencyjne

### list - listy (mutable)

```
# Przykłady
```

```
lista_liczb = [1, 2, 3, 4, 5]
```

```
lista_mieszana = [1, "dwa", 3.0, True]
```

```
lista_2d = [[1, 2], [3, 4]]
```

```
# Zastosowania: kolekcje elementów, wyniki zapytań, tymczasowe przechowywanie
```

```
zakupy = ["mleko", "chleb", "jajka"]
wyniki_testu = [85, 92, 78, 90]
```

## tuple - krotki (immutable)

**Tuple (krotka)** to uporządkowana kolekcja danych, której **nie da się zmienić po utworzeniu**, idealna do bezpiecznych, stałych struktur.

```
# Przykłady
wspolrzedne = (10, 20)
kolory_rgb = (255, 128, 0)
pojedynczy_element = (5,) # UWAGA: przecinek jest konieczny!
```

```
# Zastosowania: stałe zbiory danych, zwracanie wielu wartości z funkcji
wymiary = (1920, 1080) # rozdzielczość
data_urodzenia = (1990, 5, 15)
```

## range - zakresy

**range** to **typ sekwencyjny**, który reprezentuje **zakres liczb**. **Nie tworzy od razu listy liczb**. Generuje je „w locie” (jest wydajny pamięciowo). Najczęściej używany w pętlach **for**

```
# Przykłady
zakres1 = range(5)      # 0, 1, 2, 3, 4
zakres2 = range(1, 6)   # 1, 2, 3, 4, 5
zakres3 = range(0, 10, 2) # 0, 2, 4, 6, 8
```

```
# Zastosowania: iteracje w pętlach, generowanie indeksów
for i in range(3):
    print(f'Powtórzenie {i}')
```

```
indeksy = list(range(len(["a", "b", "c"]))) # [0, 1, 2]
```

## 4. Typy mapujące

### dict - słowniki

**Mapowanie** = przyporządkowanie wartości do **unikalnego klucza**. W Pythonie typ mapujący to **dict**. Każdy klucz **może wystąpić tylko raz**. Wartości mogą się powtarzać i mogą być **dowolnego typu**

```
# Przykłady
student = {
    "imie": "Jan",
    "nazwisko": "Kowalski",
    "wiek": 21,
    "oceny": [4.5, 5.0, 4.0]
}
```

```
pusty_sloownik = {}
sloownik_z_kluczami = dict(a=1, b=2)
```

```
# Zastosowania: konfiguracje, bazy danych w pamięci, mapowania
konfiguracja = {
    "host": "localhost",
    "port": 8080,
```

```
"debug": True
}

ksiazka_telefoniczna = {
    "Anna": "123-456-789",
    "Jan": "987-654-321"
}
```

## 5. Typy zbiorów

### set - zbiory (mutable, unikalne elementy)

```
# Przykłady
zbior_liczb = {1, 2, 3, 3, 2} # {1, 2, 3}
zbior_mieszany = {"a", 1, 3.14, True}
```

```
# Zastosowania: usuwanie duplikatów, operacje matematyczne na zbiorach
unikalne_slowa = set(["kot", "pies", "kot", "ptak"]) # {"kot", "pies", "ptak"}
```

### frozenset - zamrożone zbiory (immutable)

frozenset to: **niemodyfikowalny (immutable) zbiór**. Działa podobnie do **set**. Przechowuje **unikalne elementy** oraz podobnie **działa jak tuple**. Frozenset **nie można go zmieniać**.

#### Różnica: set vs frozenset

##### set – można zmieniać

```
s = {1, 2, 3}
s.add(4)    # OK
s.remove(2) # OK
```

##### frozenset – NIE można zmieniać

```
fs = frozenset([1, 2, 3])
```

```
fs.add(4)    # AttributeError
fs.remove(2) # AttributeError
```

```
# Przykłady
zamrozony_zbior = frozenset([1, 2, 3, 3, 2]) # frozenset({1, 2, 3})
```

```
# Zastosowania: klucze w słownikach, stałe zbiory
klucze_slownika = {
    frozenset([1, 2]): "wartość",
    frozenset(["a", "b"]): "inna wartość"
}
```

## 6. Typy logiczne

## bool - wartości logiczne

```
# Przykłady
prawda = True
fałsz = False
wynik_porownania = 10 > 5 # True
```

```
# Zastosowania: warunki, flagi, stany
jest_zalogowany = True
ma_dostep = False
```

## 7. Typy binarne

### bytes - bajty (immutable)

bytes to: **niemodyfikowalna (immutable) sekwencja bajtów**. Każdy bajt ma wartość **0–255**. Czyli nie tekst, nie liczby, **surowe dane binarne**

```
# Przykłady
bajty = b'hello'
bajty_od_listy = bytes([65, 66, 67]) # b'ABC'
```

```
# Zastosowania: dane binarne, pliki, sieć
dane_plikowe = b'\x89PNG\r\n\x1a\n' # nagłówek PNG
```

### bytearray - tablice bajtów (mutable)

bytearray to: **modyfikowalna (mutable) sekwencja bajtów**. Każdy element: **liczba 0–255** („bajty do edycji”)

```
# Przykłady
modyfikowalne_bajty = bytearray(b'hello')
modyfikowalne_bajty[0] = 72 # H zamiast h -> b'Hello'
```

```
# Zastosowania: modyfikacja danych binarnych, bufory
bufor = bytearray(1024) # bufor 1KB
```

## memoryview - widok pamięci

memoryview to: **widok na istniejący blok pamięci, w którym nie można kopiowania danych**. Działa na: bytes, bytearray, array, numpy itd. To **nie są dane** – to **okno na dane**.

```
# Przykłady
bajty = b'abcdef'
widok = memoryview(bajty)
print(widok[1:4].tobytes()) # b'bcd'
```

```
# Zastosowania: efektywna praca z dużymi danymi binarnymi
```

## 8. Specjalne typy

### NoneType - wartość None

None to: **specjalna wartość oznaczająca „brak wartości”**

```
x = None
print(type(x)) # <class 'NoneType'>
```

None **to nie 0, nie "" i nie False**. Zastosowanie, kiedy coś **nie istnieje**, coś **jeszcze nie zostało ustawione**, funkcja **nic nie zwraca**, operacja **nie ma sensu**

```
# Przykłady
brak_wartosci = None
domyslna = None

# Zastosowania: reprezentacja braku wartości, domyślne argumenty
def znajdz_uzytkownika(id):
    if id in baza_danych:
        return baza_danych[id]
    return None
```

## 9. Typy z modułów

### Decimal - liczby dziesiętne (dokładne)

```
from decimal import Decimal
```

```
# Przykłady
cena = Decimal('19.99')
podatek = Decimal('0.23')
kwota = cena * (1 + podatek)
```

*# Zastosowania: obliczenia finansowe (unikamy błędów zaokrągleń)*

### Fraction - ułamki zwykłe

```
from fractions import Fraction
```

```
# Przykłady
polowa = Fraction(1, 2)
trzecia = Fraction(1, 3)
suma = polowa + trzecia # Fraction(5, 6)
```

*# Zastosowania: obliczenia symboliczne, matematyka*