Python

ZBIÓR INFORMACJI

Deklarowanie zmiennych

Python to język dynamicznie typowany. Oznacza to, że:

- Zmienne automatycznie "domyślają" się jakiego są typu przy ich utworzeniu.
- Zmienna może przechowywać różne typy podczas swojego istnienia.

Przykład

```
1  a = 123      # Zmienna 'a' przechowuje liczbę całkowitą
2  b = 'Hello' # Zmienna 'b' przechowuje tekst
3
4  b = a
5  # b przechowuje teraz liczbę całkowitą
6  # a i b mają taką samą wartość: 123
```

Konwersje typów

Aby zmienić typ zmiennej należy użyć odpowiedniej funkcji.
W Pythonie można wyróżnić kilka najważniejszych funkcji do konwersji podstawowych typów:

```
1  int('123') # int(...) Zamienia tekst lub liczbę rzeczywistą w liczbę całkowitą
2  float('4.32') # float(...) Zamienia tekst lub liczbę całkowitą w liczbę rzeczywistą
3  str(123) # str(...) Zamienia obiekt (różnego typu) w tekst
4
5  chr(97) # chr(...) Zamienia liczbę całkowitą na odpowiadający jej znak zgodnie z ASCII
6  ord('a') # ord(...) Zamienia znak na odpowiadającą mu liczbę całkowitą zgodnie z ASCII
```

Przykład

```
1  a = 123
2  b = '456'
3
4  suma_liczb = a + int(b) # suma_liczb = 579
5  suma_tekstu = str(a) + b # suma_tekstu = '123456'
6
7  # 'a' w zapisie ASCII to 97
8  znak = chr(97) # znak = 'a'
9  ascii = ord('a') # ascii = 97
```

Matematyka w Pythonie

W Pythonie dozwolone są następujące operacje matematyczne:

```
2 + 2  # = 4   # Dodawanie
4 - 3  # = 1   # Odejmowanie
4 * 6  # = 24  # Mnożenie
32 % 5 # = 2   # Reszta z dzielenia
9 / 5  # = 1.8 # Dzielenie rzeczywiste
9 // 5 # = 1   # Dzielenie całkowite
2 ** 4 # = 16  # Potęgowanie
```

```
zmienna += 2  # Dodanie wartości
zmienna -= 3  # Odjęcie wartości
zmienna *= 6  # Pomnożenie wartości
zmienna /= 5  # Podzielenie wartości
zmienna //= 5 # Podzielenie całkowite wartości
zmienna **= 4 # Spotęgowanie wartości
```

Bardziej złożone operacje matematyczne można importować z wbudowanej biblioteki 'math':

```
from math import sqrt, sin, cos, floor, ceil, pi

abs(-12.3) # zwraca wartość bezwzględną -> 12.3
sqrt(4) # zwraca pierwiastek liczby -> 2.0
sin(pi) # zwraca wartość funkcji sin dla liczby -> 0.0
cos(pi) # zwraca wartość funkcji cos dla liczby -> -1.0
floor(5.2) # zwraca liczbę zaokrągloną w dół -> 5.0
ceil(5.2) # zwraca liczbę zaokrągloną w górę -> 6.0
round(2.37) # zwraca liczbę zaokrągloną zgodnie z matematyką -> 2.0
round(2.37, 1) # może też zaokrąglić do określonego miejsca po przecinku -> 2.4
```

Instrukcje wyjścia

Instrukcje wyjścia wyświetlają tekst w konsoli. W pythonie używa się do tego funkcji print(..):

```
1  zmienna = 123
2
3  print('Hello World!') # Wypisuje podany tekst -> Hello World
4  print(zmienna) # Wypisuje wartość zmiennej -> 123
5  print('Wartosc =', zmienna) # Wypisuje tekst i wartość zmiennej -> Wartosc = 123
```

Jest alternatywny sposób wypisywania zmiennych który można łatwiej dopasować do swoich potrzeb:

```
1  rok = 2023
2  miesiac = 9
3
4  # Dodanie znaku f przed początkiem tekstu umożliwia umieszczanie w nim wartości zmiennych
5  print(f'Dzisiaj jest {miesiac} miesiac, {rok} roku!')
6  # Wypisuje -> Dzisiaj jest 9 miesiac, 2023 roku!
```

Instrukcje wejścia

Instrukcje wejścia pozwalają odczytać podany tekst z konsoli.

W pythonie używa się do tego funkcji input(..):

```
# Czeka aż użytkownik kliknie enter i wtedy odczytuje jedną linijkę z konsoli niezależnie czy jest ona pusta czy nie.
input()
input("Podaj tekst: ") # Dodatkowo wypisze na początku podany tekst.

# Aby zamienić odczytaną linijkę na liczbę całkowitą należy użyć wcześniej wspomnianej funkcji int(..)
int(input("Podaj wartość liczbową: "))

# Aby uzyskać liczbę rzeczywistą należy użyć float(..)
float(input("Podaj wartość liczbową: "))
```

Przykład

```
imie = input("Jak masz na imie? ") # Użytkownik wpisze swoje imie i program zapamięta to jako tekst
wiek = int(input("Ile masz lat? ")) # Użytkownik wpisze swój wiek i program zapamięta to jako liczbę całkowitą

print("Witaj", imie)
print("Masz", wiek, "lat!")
```

Operatory logiczne

W Pythonie wyróżniamy 3 operatory logiczne:

- and - koniunkcja, część wspólna zbiorów, "i"
- or - alternatywa, suma zbiorów, "lub"
- not - negacja, zaprzeczenie, "nie"

Przykład

```python
a = True
b = False

a and b # Prawda jeżeli a i b jest prawdziwe
a or b  # Prawda jeżeli a lub b jest prawdziwe
not a   # Odwrócona wartość a

# Można łączyć powyższe operacje
not (a and b) # Prawda jeżeli (a i b) jest nieprawdziwe
(a and b) or not b # Prawda jeżeli (a i b) jest prawdziwe lub b jest nieprawdziwe
```

Operatory relacji

W Pythonie wyróżniamy 6 operatorów relacji:

- == - Równy
- != - Różny
- > - Większy
- < - Mniejszy
- >= - Większy lub równy
- <= - Mniejszy lub równy

Przykład

```
1  a = 12
2  b = 33
3
4  a == b # a równa się b - Nie prawda
5  a != b # a różne od b - Prawda
6  a > b  # a większe od b - Nie prawda
7  a < b  # a mniejsze od b - Prawda
8  a >= b # a większe lub równe b - Nie prawda
9  a <= b # a mniejsze lub równe b - Prawda
```

Instrukcje warunkowe

W Pythonie instrukcje warunkowe deklaruje się w następujący sposób:

```
if warunek:
    # Kod który się wykona jeśli warunek będzie spełniony, wcięcie 1 tab lub 4 spacje jest wymagane
```

Warunkiem może być operacja logiczna, operacja relacji, wartość prawda/fałsz lub ich połączenie.

Istnieje instrukcja 'elif' która sprawdzi swój warunek jeśli poprzedzający 'if' okaże się nieprawdziwy. Następnie instrukcja 'else' wykona się jeśli każdy poprzedzający 'if' i 'elif' okaże się nieprawdziwy.

Przykład

```
1  a = 12
2  b = 33
3
4  if a < 0:
5      print('a jest większe od zera')
6  elif a + b >= 100 or b < 10:
7      print('a + b jest wieksze lub rowne 100 albo b jest mniejsze od 10')
8  elif b - a == 21 and a > 5:
9      print('b - a równa się 21 i a jest większe od 5')
10 else:
11     print('Zaden z powyzszych warunkow nie byl prawdziwy')
```

Pętla for

Pętla for iteruje przez kolekcję elementów (lista / zbiór / słownik) i wykonuje kod dla każdego z nich.

Składnia

```
for element in kolekcja:
    # Kod do wykonania
```

Przykład

```
1  liczby = [5, 102, 513, 145, 128, 2048, 4]
2
3  # Wypisze wszystkie elementy z listy:
4  for element in liczby:
5      print(element)
```

Przykład

```
1  # Wypisze liczby od 0 do 99
2  # range(początek, koniec) zwraca przedział liczb całkowitych lewostronnie domknięty <początek, koniec)
3  for i in range(0, 100):
4      print(i)
```

Pętla while

Pętla while wykonuje polecenia dopóki warunek będzie spełniony.
Warunek jest sprawdzany przed każdą iteracją pętli.

Składnia

```
while warunek:
    # Kod do wykonania ...
```

Przykład

```
1  a = 1
2
3  # Wypisze potęgi 2 mniejsze lub równe 2048
4  while a <= 2048:
5      print(a)
6      a *= 2
```

Przykład

```
1  a = 10
2
3  # Wypisze liczby całkowite od 10 do 80
4  while a <= 80:
5      print(a)
6      a += 1
```

Listy

Klasa **list** przechowuje dane liniowo (ciąg danych "jedno po drugim") i pozwala na modyfikowanie elementów.

Jest indeksowana liczbowo zaczynając od 0.

Inicjowanie listy

```
1  pustaLista = [] # Inicjowanie pustej listy.
2  liczby = [81, 123, 81, 512] # Inicjowanie listy z 4 liczbami całkowitymi.
3  mieszane = ["Witam", 9,  "Cieszyn", 2023] # Inicjowanie listy z 4 róznymi elementami.
4  # Można również przechowywać listy w listach
```

Indeksowanie listy

```
1  liczby = [81, 123, 81, 512] # Inicjowanie listy z 4 liczbami całkowitymi.
2
3  a = liczby[0] # Pierwszy element - 81
4  b = liczby[3] # Czwarty element - 512
5
6  liczby[2] = 52 # Modyfikowanie trzeciego elementu z 81 na 52
```

Listy - Funkcje

Funkcje operujące na listach i zbiorach:

```python
liczby = [81, 123, 81, 512] # Inicjowanie listy z 4 liczbami całkowitymi.

posortowane = sorted(liczby) # sorted(lista) - Zwraca posortowaną rosnąco listę nie modyfikując orginalnej listy.

sumaElementow = sum(liczby) # sum(lista) - Zwraca sumę wszystkich elementów z listy.
najwiekszyElement = max(liczby) # max(lista) - Zwraca największy element z listy.
najmniejszyElement = min(liczby) # min(lista) - Zwraca najmniejszy element z listy.

dlugoscListy = len(liczby) # len(obiekt) - Zwraca ilość elementów (długość) obiektu z policzalną liczbą elementów (lista, zbiór, tekst, itp.)
```

Metody klasy **list**:

```python
liczby = [81, 123, 81, 512] # Inicjowanie listy z 4 liczbami całkowitymi.
liczby.append(2023) # .append(element) - Dodaje na koniec listy podany nowy element.
liczby.insert(2, 999) # .insert(indeks, element) - Dodaje element do listy na miejsce podanego indeksu.
# 'liczby' to teraz: [81, 123, 999, 81, 512, 2023]

liczby.sort() # .sort() - Posortuje listę rosnąco.
liczby.sort(reverse=True) # Posortuje listę malejąco.
# (Argument 'reverse' podstawowo przyjmuje wartość False lecz można podać własną wartość)

liczby.reverse() # .reverse() - Odwraca listę (ostatni element staje się pierwszym itd.)
liczby.count(81) # .count(element) - Zwraca ilość wystąpień podanego elementu w liście.
liczby.index(123) # .index(element) - Zwraca indeks pierwszego znalezionego wystąpienia elementu w liście.

liczby.pop(2) # .pop(indeks) - Usuwa element na miejscu podanego indeksu z listy.
liczby.remove(512) # .remove(element) - Usuwa pierwsze wystąpienie podanego elementu z listy.
liczby.clear() # .clear(element) - Usuwa wszystkie elementy z listy.
```

Zbiory

Klasa **set** przechowuje dane które się nie powtarzają (Konkretny element może wystąpić w zbiorze tylko raz).

Przykład

```python
zbior = set() # Inicjowanie pustego zbioru.
liczby = {5, 6, 5, 123, 5, 123, 5} # Inicjowanie zbioru z 3 elementami (duplikaty same zostaną usunięte)

liczby.add(512) # .add(element) - Dodaje element do zbioru jeśli jeszcze go tam nie ma.
liczby.add(512) # Element nie zostanie dodany drugi raz bo istnieje już w zbiorze.

liczby.remove(5) # .remove(element) - Usunie podany element ze zbioru.

iloscElementow = len(liczby) # len(zbior) - Zwraca ilość wszystkich elementów ze zbioru.
# iloscElementow = 3

# Sprawdzanie czy element występuje w zbiorze, liście lub słowniku wykonuje się operatorem 'in':
if 512 in liczby:
    print("Liczba 512 znajduje się w zbiorze 'liczby'")
```

Słowniki

Klasa **dict** przechowuje dane i jest indeksowana liczbami i tekstem (lub też niektórymi specjalnymi typami).

Zamiast **indeksów** jak w liście, występują tu **klucze**, np. "Cieszyn", "Warszawa" i "Gdańsk".

Dla słownika **każdy klucz jest unikalny** – to znaczy, że klucz nie może się powtórzyć i każdy klucz przechowuje swoją własną wartość. (Wartości mogą się powtórzyć)

Przykład

```
1  mieszkancy = {}
2
3  # Słownik można indeksować liczbami lub tekstem
4  # (jest też możliwość używania niektórych innych typów)
5
6  # Dodawanie nowych wartości do słownika:
7  mieszkancy["Cieszyn"] = 33485
8  mieszkancy["Warszawa"] = 1779304
9  mieszkancy["Gdansk"] = 486345
10
11 # Modyfikowanie wartości ze słownika:
12 mieszkancy["Gdansk"] *= 2
13
14 # Sprawdzanie czy podany klucz jest w słowniku:
15 if "Krakow" in mieszkancy:
16     print(mieszkancy["Krakow"])
17 else:
18     print("Nie ma Krakowa w słowniku!")
19
20 print(mieszkancy["Cieszyn"])
21 print(mieszkancy["Warszawa"])
22 print(mieszkancy["Gdansk"])
```

Tuple

Klasa **tuple** działa jak klasa **list** za wyjątkiem możliwości modyfikowania, dodawania i usuwania elementów - to znaczy, że elementy jak i ich ilość są nienaruszalne po utworzeniu **tupla**. Zaletą jest to, że w przeciwieństwie do list można je dodawać jako elementy do zbiorów.

Przykład

```
1  liczby = (14, 551, 65536, 123)
2
3  # a = 551 + 123 = 674
4  a = liczby[1] + liczby[3]
5
6  print("Ile liczb?", len(liczby))
7  print(liczby[0])
```

Przykład

```
1  a = 13
2  b = 2
3
4  # Wypisze: "13 2"
5  print(a, b)
6
7  # Zamiana wartości dwóch zmiennych
8  (a, b) = (b, a)
9
10 # Wypisze: "2 13"
11 print(a, b)
```

Tupli można użyć do szybkiej zamiany dwóch zmiennych.

Operacje na plikach

Odczyt

```python
plik = open("plik.txt") # open(nazwa, tryb) - Otwiera plik w podanym trybie (podstawowo tryb odczytu).

lista = plik.readlines() # .readlines() - Zwraca wszystkie linijki z pliku w postaci listy.

# Iteruje po kolei przez linijki w pliku:
for linijka in plik:
    print(linijka)

tekst = plik.read() # .read() - Zwraca całą zawartość pliku jako jeden tekst.

plik.close() # .close() - Zamyka plik i program nie ma już do niego dostępu.
```

Zapis

```python
plik = open("plik.txt", "w") # open(nazwa, tryb) - tryb "w" oznacza "write" czyli zapis.

linijki = [
    "Pierwsza linijka",
    "A to druga linijka",
    "Kolejna to trzecia",
]

plik.write("Witam\n") # .write(tekst) - Dodaje tekst do pliku.
plik.writelines(linijki) # .writelines(lista) - Dodaje linijki tekstu do pliku.

plik.close() # .close() - Zamyka plik, zapisuje go i program nie ma już do niego dostępu.
```

Deklarowanie funkcji

Funkcje deklaruje się w następujący sposób:

```
1  def nazwa(parametry):
2      # Kod do wykonania
```

Ilość parametrów jest dowolna, może ich też nie być.

Przykład

```
1  def kwadrat(a):
2      return a * a
3
4  def trojkat(a, h):
5      return a * h / 2
6
7  def przywitajSie(imie):
8      print('Witaj', imie)
9
10 poleKwadratu = kwadrat(4) # poleKwadratu = 16
11 poleTrojkata = trojkat(4, 3) # poleTrojkata = 6.0
12
13 przywitajSie('Mateusz') # Wypisze -> Witaj Mateusz
```

Funkcje mogą zwracać wartości używając instrukcji 'return'.

Kiedy funkcja zwraca wartość kończy ona swoje działanie. Jeżeli funkcja nic nie zwraca to kończy swoje działanie kiedy dotrze do jej końca.

Wykonane przez:

- Mateusz Antkiewicz