Język skryptowy lab1

Pierwszy program

Rozpocznijmy od stworzenia krótkiego programu który wyświetla „Hello world!”. W Pythonie, użyjemy polecenia print do wyświetlenia tekstu. Poniżej przedstawiono gotowy skrypt.

```
print('Hello world!')
```

Dla porównania poniżej przedstawiono program napisany w języku Java:

```
public class HelloWorld {

  public static void main(String[] args) {
    System.out.println("Hello, World");
  }
}
```

Jak widać program napisany w Javie wykonuje dokładnie tą samą czynność a jest dużo bardziej „skomplikowany”. Zatem programowanie w Pythonie jest prostsze. Proste operacje Python ma zdolność do wykonywania obliczeń. Wprowadź obliczenia bezpośrednio do konsoli Pythona a ona zwróci odpowiedź.

```
>>> 2 + 2
4
>>> 5 - 2 + 4
7
>>> (-7 + 2) * (-4)
20
>>> 6/0
Traceback (most recent call last):
File "<pyshell#0>", line 1, in <module>
6/0
ZeroDivisionError: integer division or modulo by zero
```

> W Pythonie ostatnia linia komunikatu o błędzie informuje o rodzaju błędu:exclamation:

Operatory

Operator numeryczny	opis
x + y	suma x oraz y

Operator numeryczny	opis
x - y	różnica x oraz y
x * y	iloczyn x oraz y
x / y	iloraz x oraz y
x // y	(zaokrąglony w dół) iloraz x oraz y
x % y	reszta z dzielenia x / y
abs(x)	wartość bezwzględna x
divmod(x, y)	para (x // y, x % y)
pow(x, y)	x do potęgi y
x ** y	x do potęgi y
x += y	x = x + 1
x -= y	x = x - 1
x *= y	x = x * 1
x /= y	x = x / 1

Operator porównania	opis
x != y	rózne
x == y	równe
x > y	większe
x < y	mniejsze
x >= y	większe lub równe
x <= y	mniejsze lub równe

Wiele innych języków ma specjalne operatory, takie jak "++" jako skrót dla "x += 1". Python ich nie ma.

Operacje na łańcuchach znaków (stringach)

Przy „tworzeniu” skryptu wypisującego na ekran komunikat Hello world! tekst umieszczany był w pojedynczych apostrofach (‘text’) ten sam efekt otrzymamy również gdy napis zostanie zamknięty w cudzysłów (”text”). Gdy napis wymagał będzie użycia apostrofu (np. He’s) wystarczy przed apostrofem postawić znak \ . Wypróbuj przedstawione konstrukcje.

> Backslash może być również użyty przy tabulatorach, przejściu do następnej linii, dowolnych znakach Unicode i różnych innych rzeczy, których nie można w rzeczywisty sposób wydrukować.

Do konkatenacji służy operator `+` podczas łączenia łańcuchów znaków nie ma znaczenia czy zostały one utworzone w pojedynczym czy podwójnym cudzysłowie. Łańcuchy znaków można również pomnożyć przez liczbę całkowitą np. `”spam”*4` sprawdź jaki będzie wynik takiego działania. Przydatny może być również operator indeksowania który przedstawiono w poniższym skrypcie. Wykonaj go w środowisku Pythona i sprawdź rezultat.

```
a = "Welcome to Python's world!"
print(a[0])
print(a[0:7])
```

Wprowadzanie danych

Przydatną funkcją jest możliwość wprowadzania danych przez użytkownika do tego służy instrukcja `input` i ma ona postać:

```
input("stosowny komunikat").
```

Konwertowanie typów

Jest to zamiana zmiennej z jednego typu na zmienną innego typu np. `float(2)` zamieni liczbę całkowita na zmiennoprzecinkową.

Zmienne

Zmienne odgrywają bardzo ważną rolę w większości języków programowania, a Python nie jest wyjątkiem. zmienna umożliwia zapisanie wartości przez przypisanie jej do nazwy, którą można wykorzystać do odnoszenia się do wartości później w programie. Aby przypisać zmienną, użyj jednego znaku równości.

```
>>> x = 7
>>> print(x)
7
>>> print(x + 3)
10
>>> print(x)
7
```

Zmienna może być ponownie przypisana tyle razy, ile chcesz, aby zmienić ich wartość.

```
>>> x = 123
>>> print(x)
```

```
123
>>> x = "To jest napis"
>>> print(x + "!")
To jest napis!
```

> W Pythonie zmienne nie mają określonych typów, więc można przypisać łańcuch znaków do zmiennej, a później przypisać liczbę całkowitą do tej samej zmiennej

Obowiązują pewne ograniczenia dotyczące znaków, które mogą być używane w nazwach zmiennych Pythona. Dozwolone są tylko litery, cyfry i podkreślenia. Ponadto nie mogą zaczynać się od liczb. Nieprzestrzeganie tych zasad powoduje błędy.

```
to_poprawna_nazwa_zmiennej = 7
123abc = 7
SyntaxError: invalid syntax
```

> Python to język programowania z uwzględnieniem wielkości liter. Tak więc, Lastname i lastname to dwie różne nazwy zmiennych w Pythonie. Próba odniesienia do zmiennej, która nie została przypisana powoduje błąd. możesz użyć instrukcji `del`, aby usunąć zmienną, co oznacza, że odwołanie od nazwy do wartości zostanie usunięte, a próba użycia zmiennej spowoduje błąd. Usunięte zmienne mogą być ponownie przypisane do późniejszego stanu, tak jak zwykle.

```
>>> foo = "napis"
>>> foo
'napis'
>>> bar
NameError: name 'bar' is not defined
>>> del foo
>>> foo
NameError: name 'foo' is not defined
```

> Zmienne `foo` i `bar` są nazywane zmiennymi metasyntaktycznymi, co oznacza, że są one używane jako nazwy zastępcze w przykładowym kodzie, aby coś zademonstrować. `Spam` i `eggs` są kanonicznymi zmiennymi metasyntaktycznymi używanymi w Pythonie. Nawiązują one do słynnego skeczu Latającego Cyrku Monty Pythona.

Instrukcja warunkowa IF

Możesz użyć instrukcji `if`, aby wykonać fragment kodu, jeśli spełniony jest określony warunek. Jeśli wyrażenie ma wartość `True`, wykonywane są pewne instrukcje. W przeciwnym razie są one pomijane. Instrukcja `if` wygląda następująco:

```
if wyrażenie warunkowe:
  instrukcje
```

Python używa wcięć (białe znaki na początku linii), aby ograniczyć bloki kodu. Inne języki, takie jak C czy Java, używają nawiasów klamrowych, aby to osiągnąć, ale w języku Python obowiązkowe jest wcięcie; programy nie będą działać bez niego. Jak widać, każda z instrukcji w bloku instrukcji `if` powinna być poprzedzona wcięciem. Przykład:

```
if 10 > 5:
  print("10 jest większe od 5")
print("koniec skryptu")
```

Aby wykonać bardziej złożone warunki, instrukcje mogą być zagnieżdżone, jedna w drugiej. Oznacza to, że wewnętrzne instrukcje `if` są częścią instrukcji zewnętrznej. Jest to jeden ze sposobów sprawdzenia, czy spełnione są warunki wielokrotne. Przykład:

```
if num > 5:
  print("Więcej niz 5")
  if num <= 45:
    print("Wartość z przedziału (5; 45]")
```

Instrukcja `else` wykonuje instrukcję `if` i zawiera kod, który jest wywoływany, gdy instrukcja `if` zwraca wartość `False` . Podobnie jak w przypadku instrukcji `if` , kod wewnątrz bloku powinien być wcięty. Przykład:

```
x = 4
if x == 5:
  print("Tak")
else:
  print("Nie")
```

Instrukcje `if` i `else` można łączyć łańcuchowo, aby określić, która opcja w serii możliwości jest prawdziwa. Przykład:

```
num = 12
if num == 5:
  print("Numerem jest 5")
else:
  if num == 10:
    print("Numerem jest 10")
  else:
    print("Numerem nie jest 5 ani 10")
```

Instrukcja `elif` jest skrótem do użycia podczas łączenia instrukcji `if` i `else` . Seria instrukcji `if` `elif` może mieć końcowy blok `else` , który jest wywoływany, jeśli żadne z wyrażeń `if` lub `elif` nie jest prawdziwe. Przykład:

```
num = 12
if num == 5:
  print("Numerem jest 5")
elif num == 10:
  print("Numerem jest 10")
elif num == 15:
  print("Numerem jest 10")
else:
  print("Numerem nie jest 5, 10 ani 15")
```

Operatory logiczne

Służą do tworzenia bardziej skomplikowanych warunków dla instrukcji, które opierają się na więcej niż jednym warunku. Operatorami logicznymi w Pythonie są `and`, `or`, i `not`. Operator `and` bierze dwa argumenty i ocenia je jako prawda wtedy i tylko wtedy, gdy oba argumenty są prawdziwe. W przeciwnym razie wartość ta jest fałszywa. Operator `or` także bierze dwa argumenty i ocenia je jako prawda wtedy, gdy co najmniej jeden z nich jest prawdziwy. W przeciwnym razie wartość ta jest fałszywa. Operator `not` bierze tylko jeden argument i neguje go.

> Python używa słów dla swoich operatorów logicznych, podczas gdy większość innych języków używa symboli takich jak &&, || i !.

Priorytet operatorów

Jest bardzo ważną koncepcją programowania. Jest to rozszerzenie matematycznej koncepcji porządku operacji (kolejność wykonywania działań mnożenie dokonywane przed dodaniem itp.) W celu włączenia innych operatorów, takich jak operatorów logicznych. Poniższy kod pokazuje, że `==` ma wyższy priorytet niż `or`:

```
>>> False == False or True
True
>>> False == (False or True)
False
>>> (False == False) or True
```

> Kolejność operacji Pythona jest taka sama jak w przypadku normalnej matematyki: najpierw nawiasy, potem potęgowanie, potem mnożenie / dzielenie, a następnie dodawanie / odejmowanie. Poniższa tabela zawiera listę wszystkich operatorów Pythona, od najwyższego priorytetu do najniższego.

Lp.	Operator	Opis	Lp.	Operator	Opis
1	**	potęgowanie	8	<= < > >=	mniejsze bądź równe, mniejsze,

Lp.	Operator	Opis	Lp.	Operator	Opis
					większe, większe bądź równe
2	~ + -	dopełnienie binarne, unarny plus i minus	9	== !=	równe, różne
3	* / % //	mnożenie, dzielenie, reszta z dzielenia, cześć całkowita z dzielenia	10	+= %= /= //= -= *= **=	skrótowe operatory
4	+ -	dodawanie i odejmowanie	11	is, is not	operatory tożsamości
5	<< >>	przesunięcie bitowe w lewo i w prawo	12	in, not in	operatory zawierania
6	&	AND na bitach	13	not, or, and	operatory logiczne
7	^ |		XOR i OR na bitach		

Pętla while

Instrukcja `if` jest uruchamiana jednokrotnie, jeśli jej warunek jest prawdziwy, a nigdy, jeśli oceniany jest jako fałszywy. Instrukcja `while` jest podobna, z tym wyjątkiem, że może być uruchamiana więcej niż raz. Instrukcje wewnątrz niej są wielokrotnie wykonywane, o ile warunek jest prawdziwy. Po przejściu do wartości False wykonywana jest następna sekcja kodu. Poniżej znajduje się pętla `while` zawierająca zmienną, która liczy od 1 do 5, w którym to momencie pętla kończy się.

```
i = 1
while i <= 5:
  print(i)
  i = i + 1
print("Koniec!")
```

Nieskończona pętla jest specjalnym rodzajem pętli while; nigdy się nie kończy. Jego stan pozostaje zawsze prawdziwy. Przykład:

```
while 1 == 1:
  print("operacje w pętli")
```

Aby przedwcześnie zakończyć pętlę, można użyć instrukcji break. Po napotkaniu wewnątrz pętli instrukcja break powoduje natychmiastowe zakończenie pętli. Przykład:

```
i = 0
while 1 == 1:
  print(i)
  i = i + 1
  if i >= 5:
    print("Przerwanie")
    break
print("Koniec")
```

Kolejną instrukcją, która może być używana w pętlach, jest continue. W przeciwieństwie do break, continue przeskakuje z powrotem na początek pętli, zamiast ją zatrzymywać. Przykład:

```
i = 0
while True:
  i = i +1
  if i == 2:
    print("Pomiń 2")
    continue
  if i == 5:
    print("Przerwanie")
    break
  print(i)
print("Koniec")
```

Listy

Listy są innym typem obiektu w Pythonie. Służą do przechowywania indeksowanej listy pozycji. Listę tworzy się za pomocą nawiasów kwadratowych z przecinkami oddzielającymi elementy. Dostęp do określonej pozycji na liście można uzyskać za pomocą jej indeksu w nawiasach kwadratowych.

> Listy indeksujemy od 0. Przykład:

```
words = ["Hello", "world", "!"]
print(words[0])
print(words[1])
print(words[2])
```

Pusta lista tworzona jest przez pustą parę nawiasów kwadratowych. Zazwyczaj lista będzie zawierała elementy jednego typu, ale możliwe jest również uwzględnienie kilku różnych typów. Listy można również zagnieżdżać w innych listach. Przykład:

```
number = 3
things = ["string", 0, [1, 2, number], 4.56]
```

```
print(things[1])
print(things[2])
print(things[2][2])
```

> Listy są często używane do reprezentowania siatek 2D, ponieważ w Pythonie brakuje wielowymiarowych tablic, które byłyby używane w innych językach.

Indeksowanie wykraczające poza granice list powoduje błąd `IndexError`. Niektóre typy, takie jak łańcuchy znaków, mogą być indeksowane jak listy. Indeksowane łańcuchy zachowują się tak, jak indeksowana lista przechowująca każdy znak osobno. W przypadku innych typów, takich jak liczby całkowite, indeksowanie ich nie jest możliwe i powoduje błąd `TypeError`.

###Operacje na listach Elementowi listy o określonym indeksie można ponownie nadać nową wartość .

Przykład:

```
nums = [7, 7, 7, 7]
nums[2] = 5
print(nums)
```

Listy mogą być łączone i mnożone ze sobą tak samo łączone jak stringi. Przykład:

```
nums = [1, 2, 3]
print(nums + [4, 5, 6])
print(nums * 3)
```

Aby sprawdzić, czy dany element znajduje się na liście, można użyć operatora `in`. Zwraca `True`, jeśli element występuje raz lub więcej razy na liście, a `False`, jeśli nie występuje. Przykład:

```
words = ["spam", "egg"]
print("spam" in words)
print("sausages" in words)
```

Aby sprawdzić, czy dany element nie znajduje się na liście, możesz użyć operatora `not` w jeden z następujących sposobów: Przykład:

```
nums = [1, 2, 3]
print(not 4 in nums)
print(4 not in nums)
print(not 3 in nums)
print(3 not in nums)
```

Funkcje list

Innym sposobem modyfiakcji list jest użycie metody `append`. Spowoduje to dodanie elementu do końca istniejącej listy. Przykład:

```
nums = [1, 2, 3]
nums.append(4)
print(nums)
```

Aby pobrać liczbę elementów w liście można użyć funkcji `len`. Przykład:

```
nums = [4, 5, 6]
print(len(nums))
```

> W przeciwieństwie do `append`, `len` jest normalną funkcją, a nie metodą. Oznacza to, że jest ona napisana przed listą, dla której jest wywoływana, bez użycia kropki.

Metoda `insert` jest podobna do `append`, z tym wyjątkiem, że pozwala wstawić nowy element w dowolnej pozycji na liście, a nie tylko na końcu. Przykład:

```
words = ['Python', 'cool']
words.insert(1, 'is')
print(words)
```

Metoda `index` znajduje pierwsze wystąpienie elementu listy i zwraca jego indeks. Jeśli element nie znajduje się na liście, wywołuje on błąd `ValueError`. Przykład:

```
letters = ['q', 'w', 'e', 'r', 't', 'y']
print(letters.index('r'))
print(letters.index('a'))
```

Istnieje jeszcze kilka przydatnych funkcji i metod dla list. `max(lista)` Zwraca pozycję listy z maksymalną wartością `min(lista)` Zwraca pozycję listy z minimalną wartością `lista.count(obj)` Zwraca liczbę określającą, ile razy element występuje na liście `lista.remove(obj)` Usuwa obiekt z listy `lista.reverse()` Odwraca obiekty na liście `lista.pop(i)` Zwraca i-ty element listy i usuwa go `lista.extend(seq)` Dołącza zawartość seq do listy `lista.sort(func)` Sortuje obiekty z listy, użyj funkcji porównawczej func

Pętla for

Pętla `for` iteruje po elementach np. listy w kolejności, wykonując blok instrukcji. Przykład:

```
words = ["hello", "world", "spam", "eggs"]
for word in words:
  print(word + "!")
```

Pętla `for` jest zwykle używana do powtarzania fragmentu kodu pewną liczbę razy. Odbywa się to przez połączenie pętli z funkcją `range`. Przykład:

```
for i in range(5):
  print(i)
```

Zadania do wykonania

:one: Przetestuj kilka podstawowych obliczeń (+, -, *, /). Spróbuj wykonać kilka różnych obliczeń z różnymi operatorami.

:two: Napisz skrypt który zapyta użytkownika o imię a potem wyświetli powitanie używając z wykorzystaniem podanego imienia.

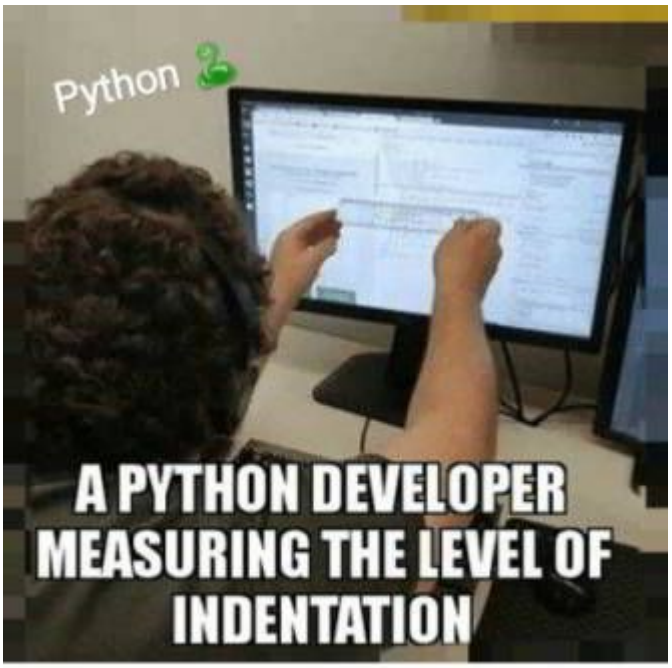
:three: Napisz krótki skrypt (wystarczy jedna linia kodu) który obliczy sumę liczb całkowitych wprowadzonych przez użytkownika i wynik wyświetli jako liczba zmiennoprzecinkowa.

:four: Korzystając z funkcji `range` oraz `sum` oblicz sumę liczb naturalnych od 8 do 80. (Skorzystaj z dokumentacji dla tych funkcji).

:five: Napisz skrypt, który wylicza na podstawie zadanego argumentu (daty w postaci rrrr-mm-dd) liczbę dni od tego czasu do daty aktualnej. Skorzystaj z: https://docs.python.org/3/library/datetime.html.

:six: Korzystając ze zdobytych wiadomości napisz prosty kalkulator posiadający menu wyboru działania i wykonującego działania dla dwóch liczb.

:exclamation: zadania 2-6 mają zostać dodane na GitHuba :exclamation:



Indentation checking in Python is like: