Wprowadzenie do instrukcji Pythona

- *Instrukcje* (ang. statement) pozwalają na przekazanie Pythonowi tego co mają robić nasze programy. Łącząc instrukcje określamy procedurę wykonywaną przez Pythona w celu zrealizowania programu.
- Innym sposobem zrozumienia roli instrukcji jest powrócenie do hierarchii wprowadzonej w poprzedniej części wykładu:

✓Program składa się z modułów

✓Moduły zawierają instrukcje

✓*Instrukcje zawierają wyrażenia*

✓Wyrażenia tworzą i przetwarzają obiekty

- Instrukcje wykorzystuję i kierują wyrażenia do przetwarzania obiektów omawianych w poprzedniej części wykładu
- W instrukcjach obiekty zaczynają istnieć (np. instrukcja przypisania tworzy nowy obiekt)

Wprowadzenie do instrukcji Pythona

- Przegląd instrukcji Pythona (źródło: *Python. Wprowadzenie. Wydanie IV Autor: Mark Lutz*)

Instrukcja	Rola	Przykład
Przypisanie	Tworzenie referencji	`a, *b = 'dobry', 'zły', 'paskudny'`
Wywołania	Wykonywanie funkcji	`log.write("mielonka, szynka")`
Wywołania `print`	Wyświetlanie obiektów	`print('The Killer', joke)`
`if/elif/else`	Wybór działania	`if "python" in text:` `  print(text)`
`for/else`	Iteracja po sekwencjach	`for x in mylist:` `  print(x)`
`while/else`	Ogólne pętle	`while X > Y:` `  print('witaj')`
`pass`	Pusty pojemnik	`while True:` `  pass`
`break`	Wyjście z pętli	`while True:` `  if exittest(): break`
`continue`	Kontynuacja pętli	`while True:` `  if skiptest(): continue`
`def`	Funkcje i metody	`def f(a, b, c=1, *d):` `  print(a+b+c+d[0])`

Wprowadzenie do instrukcji Pythona

- Przegląd instrukcji Pythona (źródło: *Python. Wprowadzenie. Wydanie IV Autor: Mark Lutz*) c.d.

Instrukcja	Rola	Przykład
`return`	Wynik funkcji	`def f(a, b, c=1, *d):` `  return a+b+c+d[0]`
`yield`	Funkcje generatora	`def gen(n):` `  for i in n: yield i*2`

`global`	Przestrzenie nazw	`x = 'stary'`
		`def function():`
		`  global x, y; x = 'nowy'`

`nonlocal`	Przestrzenie nazw (3.0+)	`def outer():`
		`  x = 'stary'`
		`  def function():`
		`  nonlocal x; x = 'nowy'`

`import`	Dostęp do modułów	`import sys`
`from`	Dostęp do atrybutów	`from sys import stdin`

`class`	Budowanie obiektów	`class Subclass(Superclass):`
		`  staticData = []`
		`  def method(self): pass`

Wprowadzenie do instrukcji Pythona

- Przegląd instrukcji Pythona (źródło: *Python. Wprowadzenie. Wydanie IV Autor: Mark Lutz*) c. d.

Instrukcja	Rola	Przykład

`try/except/finally`	Przechwytywanie wyjątków	`try:`
		`  action()`
		`except:`
		`  print('Błąd w akcji')`

`raise`	Wywoływanie wyjątków	`raise endSearch(location)`
`assert`	Sprawdzanie w debugowaniu	`assert X > Y, 'X jest za małe'`
`with/as`	Menedżery kontekstu (2.6+)	`with open('data') as myfile:` `  process(myfile)`

`del`	Usuwanie referencji	`del dane[k]`
		`del dane[i:j]`
		`del obiekt.atrybut`
		`del zmienna`

Wprowadzenie do instrukcji Pythona

- Uwagi dotyczące składni instrukcji języka Python:
- Rozważmy dla przykładu instrukcję warunkową *if* zakodowaną w języku C:

```
if (x > y) {
 x = 1;
 y= 2;
}
```

I tą samą instrukcję w języku Python

```
if  x > y:
    x = 1
    y = 2
```

- Po pierwsze instrukcja w Pythona używa mniej elementów składniowych.
- Tym jednym elementem składniowym który dodaje Python jest znak dwukropka (:)

Wprowadzenie do instrukcji Pythona

- Wszystkie instrukcje złożone w Pythonie (czyli instrukcje z zagnieżdżonymi kolejnymi instrukcjami) piszemy zgodnie z jednym wzorcem – z nagłówkiem zakończonym dwukropkiem, po którym następuje zagnieżdżony blok instrukcji, ***wcięty*** w stosunku do wiersza nagłówka

Wiersz_nagłówka**:**

Instrukcja_1

Instrukcja_2

- *Nawiasy są opcjonalne*: np. nawiasy wokół testów możemy pominąć a instrukcja nadal będzie działać poprawnie

if (x>y) if x>y

- *Koniec wiersza jest końcem instrukcji.* W Pythonie nie trzeba kończyć instrukcji średnikiem

x=1; *x=1*

Wprowadzenie do instrukcji Pythona

- *Koniec wcięcia to koniec bloku:*

W Pythonie wcina się wszystkie instrukcje zagnieżdżone o tą samą odległość w prawo. Python wykorzystuje fizyczne podobieństwo instrukcji do ustalenia, gdzie blok się zaczyna i gdzie kończy

Przez *indetancję* rozumiemy puste białe znaki znajdujące się po lewej stronie zagnieżdżonych instrukcji. Pustymi znakami mogą być spacje i tabulatory. Nie ma znaczenia również ich ilość. Reguła składni mówi jedynie, że w jednym bloku zagnieżdżonym wszystkie instrukcje muszą być zagnieżdżone na tą samą odległość w prawo. Jeśli tak nie jest otrzymamy błąd składni

Reguła indetancji wymusza w programistach tworzenie jednolitego, regularnego i czytelnego kodu, rezultatem jest bardziej spójny i czytelny kod.

Nie należy jednak z zasady w jednym bloku mieszać tabulatorów i spacji

Wprowadzenie do instrukcji Pythona

- Przykład demonstrujący często spotykane błędy indentancji (źródło: *Python. Wprowadzenie. Wydanie IV Autor: Mark Lutz*)

```
   x = 'MIELONKA'                    # Błąd — wcięcie pierwszego wiersza
if 'płot' in 'żywopłot':
   print(x * 8)
      x += 'NI'                      # Błąd — nieoczekiwana indentacja
      if x.endswith('NI'):
            x *= 2
         print(x)                    # Błąd — niespójna indentacja
```

```
x = 'MIELONKA'
if 'płot' in 'żywopłot':
   print(x * 8)
   x += 'NI'
   if x.endswith('NI'):
      x *= 2
      print(x)                       # Wyświetla "MIELONKANIMIELONKANI"
```

Wprowadzenie do instrukcji Pythona

- *Instrukcje if*

Kiedy instrukcja if jest wykonywana Python wykonuje blok kodu powiązany z *pierwszym* testem zwracającym wynik będący *prawdą* lub blok *else* jeśli wszystkie testy zwracają wynik będący fałszem. Ogólna forma instrukcji *if* przedstawia się w następujący sposób:

if <test1>**:**
        <instrukcje1>
elif <test2>**:**
        <instrukcje1>
else:
        <instrukcje3>

Wprowadzenie do instrukcji Pythona

Testy prawdziwości, operator Boolean

- Porównania i testy równości: **==, !=, >,<, >=, <=** zwracają True lub False (odpowiednik liczb 1 i 0)
- Dowolna liczba niebędąca zerem i dowolny niepusty obiekt są prawdą
- Liczby o wartości zero, puste obiekty uznawane są za fałsz

Operatory Boolean *and* i *or* wykorzystuje się do łączenia wyników testów. W Pythonie istnieją trzy operatory wyrażeń Boolean

X and Y

jest prawdziwe kiedy zarówno X, jak i Y jest prawdziwe

X or Y

jest prawdziwe, kiedy X lub Y jest prawdziwe

not X

jest prawdziwe, kiedy X jest fałszywe

Wprowadzenie do instrukcji Pythona

Wyrażenie trójargumentowe if/else

- Jedną z często spotykanych ról operatorów Boolean jest zapisywanie w kodzie wyrażeń działających tak samo jak instrukcje if. Rozważmy instrukcję ustawiającą zmienną A na Y w zależności od wartości prawdy X

```
if X:
        A = Y
else:
        A = Z
```

Możemy zapisać powyższą instrukcję w jednym wyrażeniu

```
A = Y if X else Z
```

Wyrażenie to ma dokładnie ten sam efekt, podobny efekt uzyskamy po uważnym połączeniu operatorów and i or

```
A = ((X and Y) or Z)
```

Wykład 3

Wprowadzenie do instrukcji Pythona

Pętle while, for

- Pętla - specjalna konstrukcja językowa, która ma za zadanie powtarzać pewien zestaw instrukcji określoną ilość razy
- iteracja - pojedyncze wykonanie pętli
- iterator - licznik przechowujący numer właśnie wykonanej iteracji
- Instrukcja *while* jest najbardziej uniwersalną konstrukcją iteracyjną tego języka, powtarza wykonanie bloku instrukcji, dopóki test znajdujący się w nagłówku zwraca wartość True. Kiedy test zwróci wartość False sterowanie przechodzi do instrukcji następującej po bloku while.
- Instrukcja while składa się z wiersza nagłówka z wyrażeniem testowym, bloku instrukcji oraz opcjonalnie części *else* wykonywanej , kiedy sterowanie opuszcza pętle bez napotkania instrukcji *break*.

Wprowadzenie do instrukcji Pythona

Pętle while, for

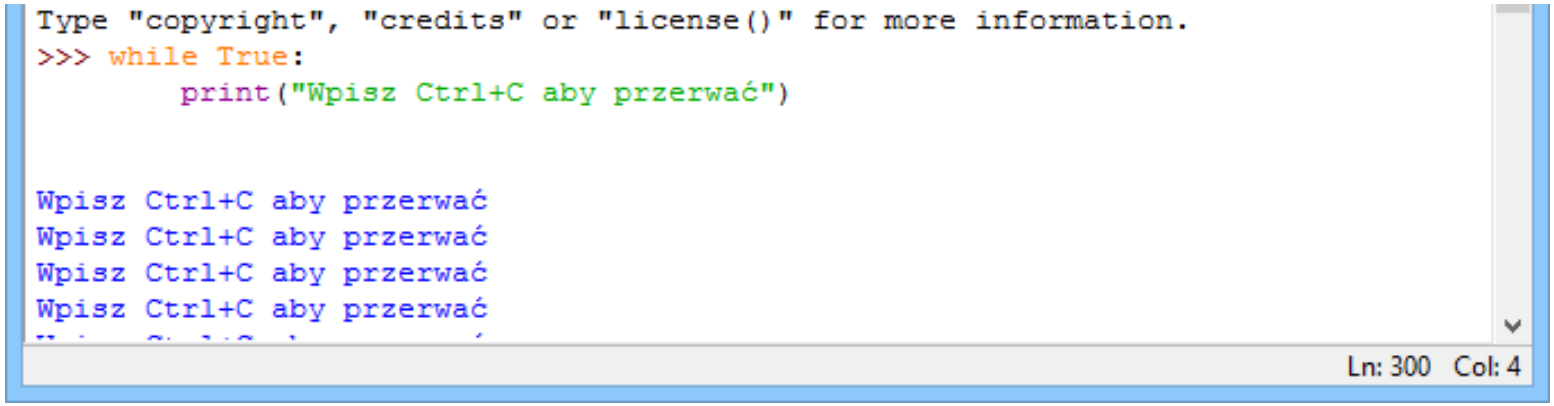
while *<test>**:***

<blok instrukcji>

else:

<blok instrukcji>

Przykład pętli nieskończonej



Wprowadzenie do instrukcji Pythona

Pętle while, for

Przykład pętli odcinającej ostatni znak łańcucha, i kodu odliczającego od a do b

```
>>> x = 'mielonka'
>>> while x:
        print(x)
        x = x[1:]


mielonka
ielonka
elonka
lonka
onka
nka
ka
a
>>>
```

```
>>> a=0
>>> b=10
>>> while a<b:
        print(a, end=' ')
        a+=1

0 1 2 3 4 5 6 7 8 9
>>>
```

Wprowadzenie do instrukcji Pythona

Pętle while, for

W Pythonie nie ma odpowiednika pętli z warunkiem na końcu. W razie konieczności możemy zastąpić ją za pomocą testu i instrukcji *break* na końcu ciała pętli

while True:

<instrukcje>

if <test>: break

break – wychodzi z najbliższej obejmującej daną instrukcję pętli (omija całą instrukcję pętli)

continue – przechodzi na górę najbliższej obejmującej daną instrukcję pętli (do jej wiersza nagłówka)

pass – pusty pojemnik instrukcji

blok pętli *else* – wykonywany jest wtedy (i tylko wtedy), gdy pętla kończy się normalnie – bez trafienia na instrukcję break

Wprowadzenie do instrukcji Pythona

Pętle while, for

Ogólny format pętli while będzie wyglądał następująco

while <test1>:

<instrukcje1>

if <test2>: break

if <test3>: continue

else:

<instrukcje1>

Instrukcje break i continue mogą się pojawić w dowolnym miejscu ciała pętli while (lub for) lecz zazwyczaj umieszczane są jako zagnieżdżone w teście if .

Wprowadzenie do instrukcji Pythona

Pętle while, for

Część *else* pętli *while* jest unikalna dla Pythona i często dla osób początkujących myląca. Udostępnia ona jawną składnię służącą do zapisu często występującego scenariusza. To struktura pozwalająca na przechwytywanie innej drogi wyjścia z pętli bez ustawiania i sprawdzania warunków opcji statusu (źródło: *Python. Wprowadzenie. Wydanie IV Autor: Mark Lutz*)

```
found = False
while x and not found:
    if match(x[0]):
        print('Ni')
        found = True
    else:
        x = x[1:]
if not found:
    print('Nie znaleziono')
```

```
while x:
    if match(x[0]):
        print('Ni')
        break
    x = x[1:]
else:
    print('Nie znaleziono')
```

Wprowadzenie do instrukcji Pythona

Pętle while, for

- Pętla for jest w Pythonie uniwersalnym iteratorem po sekwencjach. Może przechodzić elementy po dowolnym obiekcie będącym uporządkowaną sekwencją
- Pętla for rozpoczyna się od wiersza nagłówka określającego cel przypisania wraz z obiektem, który chcemy przechodzić

for <cel> **in** <obiekt>**:**

<instrukcje>

else:

<instrukcje>

- Kiedy Python wykonuje pętle for, jeden po drugim przypisuje elementy z obiektu sekwencji do celu i wykonuje dla każdego z nich ciało pętli

Wprowadzenie do instrukcji Pythona

Pętle while, for

- Instrukcja *for* obsługuje również opcjonalny blok *else*, który działa dokładnie tak samo jak w pętli while – jest wykonywany wtedy, gdy pętla kończy się bez trafienia na instrukcję *break (to znaczy przetworzone zostały wszystkie elementy sekwencji)*
- Pełny format pętli *for* z instrukcją *break i continue*

for <cel> **in** <obiekt>**:**
    <instrukcje>
    if <test>: break
    if <test>: continue
else:
    <instrukcje>

Wprowadzenie do instrukcji Pythona

Pętle while, for

Przykłady: Lista jako <obiekt> pętli for

```
#Wyświetlenie elementów listy
Lista_zakupów = ["mielonka", "jajka", "szynka"]
for x in Lista_zakupów:
        print(x, end=' ')

#Suma po elementach listy
suma = 0
for x in [1, 2, 3, 4]:
    suma+=x
print('\nSuma=',suma)

#Iloczyn z elemntów listy
iloczyn = 1
for x in [1, 2, 3, 4]:
    iloczyn*=x
print('Iloczyn=',iloczyn)
```

Wprowadzenie do instrukcji Pythona

Pętle while, for

Przykłady: W połączeniu z pętlą for jako obiekt może być użyty dowolny typ sekwencji. Pętla for działa również na łańcuchach znaków i krotkach

```
#Iteracja po elementach łańcucha
S = 'drwal'
for x in S: print(x, end=' ')
print()

#Iteracja po elementach krotki
T = ("ja", "jestem", "drwalem")
for x in T: print(x, end=' ')
```

Krotki w pętlach for przydają się także do iteracji po kluczach i wartościach słowników za pomocą metody *items*

```
#Iteracja po elementach słownika
D = {'a':1, 'b':2, 'c':3}
for klucz in D:
    print(klucz, '=>', D[klucz])

print(list(D.items()))

for (klucz, wartość) in D.items():
    print(klucz, ' : ', wartość)
```

Pętle

- Przykład pętli zagnieżdżonych z częścią else. Mając podaną listę obiektów (items) oraz listę kluczy(tests) poniższy kod wyszukuje w liście obiektów każdego klucza i zgłasza rezultat tego działania

```
items = ['aaa', 111, (4,5), 2.12]
tests = [(4,5), 3.14]
for key in tests:
    for item in items:
        if item == key:
            print(key, 'znaleziono')
            break
    else:
        print(key,' nie znaleziono')


(4, 5) znaleziono
3.14  nie znaleziono
```

- Ponieważ zagnieżdżona instrukcja if wykonuje instrukcję break, kiedy klucz zostanie dopasowany, część else zakłada, że jeśli do niej dojdzie to wyszukiwanie się powiodło. Część else jest wcięta na tę samą odległość co wiersz nagłówka wewnętrznej pętli for, czyli jest powiązana właśnie z nią

Pętle

- Innym rozwiązaniem jest użycie operatora *in*. Operator *in* w niejawny sposób przeszukuje obiekt, szukając dopasowania (przynajmniej logicznego), dzięki czemu możemy zastąpić pętle zagnieżdżone

```
items = ['aaa', 111, (4,5), 2.12]
tests = [(4,5), 3.14]

for key in tests:
    if key in items:
        print(key, 'znaleziono')
    else:
        print(key,' nie znaleziono')
```

Wprowadzenie do instrukcji Pythona

Pętle while, for

- Pętla *for* zalicza się do pętli liczonych, jest łatwiejsza do zapisania od while dlatego jest pierwszą instrukcją po którą sięgamy kiedy musimy przejść daną sekwencje.
- Python udostępnia dwie funkcje wbudowane, które pozwalają na wykonania wyspecjalizowanej pętli

✓ Wbudowana funkcja *range,* która zwraca serię kolejnych liczb całkowitych, które mogą zostać wykorzystane jako indeksy (licznik) pętli *for*

✓ Wbudowana funkcja *zip* zwracająca serie krotek równoległych elementów, która może być wykorzystana do przechodzenia wielu sekwencji w pętli *for*

Wprowadzenie do instrukcji Pythona

Pętle while, for

- Funkcja *range* jest najczęściej wykorzystywana do generowania indeksów dla pętli *for*. Można jej użyć w każdym miejscu, w którym potrzebna jest nam lista liczb całkowitych.
- W Pythonie 3.0 *range* jest *iteratorem* generującym elementy na żądanie, dlatego musimy opakować tę funkcję w wywołanie *list* w celu wyświetlenia wszystkich wyników naraz

```
#iterator range
print(list(range(5)), list(range(2,5)), list(range(0,10,2)))
```

```
[0, 1, 2, 3, 4] [2, 3, 4] [0, 2, 4, 6, 8]
```

- Z jednym argumentem funkcja *range* generuje listę liczb całkowitych od zera do wartości argumentu (bez niej samej), z dwoma argumentami pierwszy uznawany jest za dolną granicę, z trzema trzeci argument jest krokiem

Wprowadzenie do instrukcji Pythona

Pętle while, for

- Pętle for automatycznie wymuszają wyniki *range* w Pythonie 3.0, dlatego nie musimy tutaj używać wywołania *list*

```
>>> for i in range(3):
        print(i, 'Python')


0 Python
1 Python
2 Python
```

- Przykład przechodzenia niewyczerpującego – range i wyciniki

```
>>> S = 'abcdefghijk'
>>> list(range(0,len(S),2))
[0, 2, 4, 6, 8, 10]
>>> for i in range(0,len(S), 2): print(S[i],end=' ')

a c e g i k
>>> for c in S[::2]: print(c,end=' ')

a c e g i k
```

Wprowadzenie do instrukcji Pythona

Pętle while, for

- Funkcja *zip* pozwala na wykorzystanie pętli for do *równoległego* przejścia większej liczby sekwencji. Jako argumenty przyjmuje jedną lub więcej sekwencji i zwraca serie krotek łączących w pary równoległe elementy z tych sekwencji. Tak jak range, *zip* jest obiektem na którym można wykonywać iteracje, dlatego musimy wywołać tę funkcję za pomocą polecenia *list* aby wyświetlić wyniki

```
>>> L1 = [1, 2, 3, 4]
>>> L2 = [5, 6, 7, 8]
>>> list(zip(L1,L2))
[(1, 5), (2, 6), (3, 7), (4, 8)]
```

- W połączenie z pętlą for zip pozwala na *iteracje równoległe*

```
>>> for (x,y) in zip(L1,L2):
        print(x,' + ',y, ' = ',x+y)


1  +  5  =  6
2  +  6  =  8
3  +  7  =  10
```