Faculty of Geodesy and Cartography
WARSAW UNIVERSITY OF TECHNOLOGY

Informatyka Geodezyjna - wykłady/ćwiczenia, rok akad. 2021-2022

Wyk. 5: Python - pętla while

Kinga Węzka
kinga.wezka@pw.edu.pl
Katedra Geodezji i Astronomii Geodezyjnej

Warsaw University of Technology

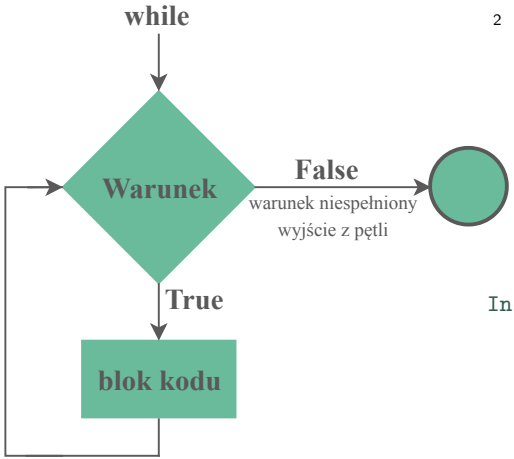
1. Sterowanie wykonywaniem programu
2. Pętle w Pythonie
3. Pętla while
4. Przykłady pętli while
   - Przykłady pętli while – pętla nieskończona – while True:
   - Przykłady pętli while – iteracja po elementach sekwencji
   - Przykłady pętli while - zbieżność rozwiązania
5. Pętle zagnieżdżone (ang. nested loops)
6. Wyrażenie else dla pętli while
7. Instrukcje kontrolne w pętli while
8. Walrus operator (wersja Python $>= 3.8$) – wyrażenie przypisania
9. Pętla do-while w Pythonie – jej odpowiednik
10. Podsumowanie

- Instrukcje warunkowe: **if/elif/else**
- Instrukcje kontrolne: **break**, **continue** i **pass**
- Pętle: **for while**

W Pythonie używane są dwa rodzaje pętli: for i while

- `for` : "przebiega" przez podany zbiór danych, element-po-elemencie. Pętli używamy jeżeli wiemy ile iteracji chcemy wykonać - znamy liczbę iteracji (nawet jeśli możesz to wcześniej zatrzymać)
- `while`: wykonuje się dopóki pewien warunek logiczny jest spełniony. Pętli używamy jeżeli nie wiemy kiedy się ona zakończy (nie znamy liczby iteracji) natomiast znamy jakiś specyficzny warunek który powinien zatrzymać iteracje.

Obie pętle są używane w celu przetwarzania powtarzającej się sekcji kodu. W przeciwieństwie do pętli **for** pętla **while** nie jest uruchamiana n-razy, ale dopóki zdefiniowany warunek nie będzie już spełniony. Jeśli warunek jest na początku (w pierwszym przejściu) fałszywy, blok kodu w pętli nie zostanie w ogóle wykonany.




```
while <warunek>: # test pętli
    blok kodu: wykonanie iteracji
```

- Sprawdzany jest warunek pętli, jeśli jest `False`, pętla zostaje zakończona, a sterowanie przeskakuje do następnej instrukcji w programie (po pętli).
- Jeśli warunek jest `True`, zestaw instrukcji wewnątrz pętli zostanie wykonany, a następnie przeskoczy na początek pętli w celu następnej iteracji.

In[1]:

```
i = 1
while i < 3:
    print(i)
    i += 1
```

Out[1]:

```
1
2
```

- Pętla **while ... True:** (tzw. pętla nieskończona) wykorzystywana jest w sytuacjach kiedy nie znamy momentu zakończenia (nie znamy liczby iteracji), np. przy pobieraniu danych wejściowych w oczekiwaniu na podanie komendy równej warunkowi stopu pętli.
- Pętla nieskończona powinna zawierać instrukcje kontrolną `break`, która przerwie jej działanie.

```
1 counter = 0
2 while True: # pętla nieskonczona
3     counter += 1
4     if counter > 10:
5         break
```

In[2]:

```
lista = []
print("Podaj liczby całkowite, które chcesz umieścić w liście.")
print("Wpisz 'stop' aby zakończyć")
while True:
    wejscie = input()
    if wejscie == 'stop':
        break
    lista.append(int(wejscie))
print("Twoja lista:" , lista)
```

Out[2]:

```
3
33
333
stop
Twoja lista: [3, 33, 333]
```

- W celu iteracji po elementach sekwencji należy wykorzystać indeksowanie, a liczbę elementów można srawdzić za pomocą metody `len()`.

In[3]:

```
L = [1, 2, 3, 4]
i = 0 # index
while i < len(L):
    print(L[i])
    i += 1
```

Out[3]:

```
1
2
3
4
```

- Gdy sekwencja typu lista jest testowana w kontekście boolowskim, to jeśli lista zawiera elementy otrzymujemy `True`, a jeśli lista jest pusta otrzymujemy `False`.
- Metoda pop pobiera element i usuwa go z listy, więc po pobraniu wszystkich elementów lista jest pusta, a jest fałszywe, i pętla się kończy.

In[4]:

```
a = ['foo', 'bar', 'baz']
while a:
    print(a.pop(-1))
```

Out[4]:

```
baz
bar
foo
```

- Iteracje w celu osiągnięcia konwergencji (zbieżność) rozwiązania.
- Zbieżność można sprawdzić sprawdzając różnicę między bieżącą wartością z, a jej poprzednią wartością. Ta różnica to wartość dodawana przy każdej iteracji, czyli fz/dfz.

In[5]:

```
n   = 9 # max. liczba iteracji
z   = 4 #
fz  = z**4 - 1 # funkcja f(z)
dfz = 4 * z**3 # pierwsza pochodna f(z)
while n > 0:
    fz = z**4 - 1       # f(z)
    dfz = 4 * z**3      # pierwsza pochodna f(z)
    z = z - fz / dfz      # metoda Newton-Raphson
    if abs(fz / dfz) < 0.25: # koniec: zbieżność osiągnięta
        break
    print('iter: ', n, round(z,3), round(fz / dfz,3) )
    n-=1
```

Zagnieżdżone pętle (ang. *nested loops*) while w Pythonie, wykorzystywane są jeżeli chcemy wykonać kilkukrotnie nieznaną liczbę iteracji z różnymi warunkami.

In[6]:

```
i = 2
while(i < 20):
    j = 2
    while(j <= (i/j)):
        if not(i % j):
            break
        j = j + 1
    if (j > i/j) :
        print(i, "- liczba pierwsza")
    i = i + 1
```

Out[6]:

```
2 - liczba pierwsza
3 - liczba pierwsza
5 - liczba pierwsza
7 - liczba pierwsza
11 - liczba pierwsza
13 - liczba pierwsza
17 - liczba pierwsza
19 - liczba pierwsza
```

Po pętli `while` można umieścić instrukcje `else`. Kod pod instrukcją `else` wykonywany jest tylko gdy pętla jest "wyczerpana". Instrukcja `else` nie wykona się gdy przerwiemy przejścia instrukcją `break` lub gdy zostanie zgłoszony wyjątek (ang. *exception raised*).

In[7]:

```
a=3
while(a>0):
    print(a)
    a-=1
else:
    print('0 osiągniete')
```

Out:

```
3
2
1
0 osiągniete
```

In[8]:

```
a=3
while(a>0):
    print(a)
    if a<3:
        break
    a-=1
else:
    print('0 osiągniete')
```

Out:

```
3
2
```

continue

Kończy przebieg aktualnej iteracji pętli i rozpoczyna kolejną iteracją.

In[9]:

```
counter = 0
while True:
    counter += 1
    if counter%2 == 0:
        continue
    if counter > 6:
        break
    print(counter)
```

Out:

```
1
2
5
```

break

Powoduje zakończenie pętli, kod po tej instrukcji w pętli nie zostanie wykonany.

In[10]:

```
counter = 0
while True:
    counter += 1
    if counter > 4:
        break
    print(counter)
```

Out:

```
1
2
3
4
```

- Dodano nową składnię i funkcjonalność operatora :=, operator ten jest nazywany **operatorem morsa** (ang. *walrus operator*) ze względu na składniowe podobieństwo do oczu i kłów morsa. Składnia:

```
NAME := expression
```

- Operator morsa umożliwia przypisanie i zwrócenie wartości w tym samym wyrażeniu. Na przykład, jeśli chcesz przypisać zmienną i wydrukować jej wartość, zwykle robisz coś takiego:

Bez operatora morsa:

In[11]:
```
walrus = False
print(walrus)
```

Out[11]:
```
False
```

Z operatorem morsa:

In[12]:
```
print(walrus := True)
```

Out[12]:
```
True
```

- Poniższy kod prosi użytkownika o podanie danych wejściowych, dopóki nie wpisze "quit". Powtarzamy `input()` i musimy dodać `current()` do listy, zanim o to zapytamy:

```
inputs = list()
current = input("Write something: ")
while current != "quit":
    inputs.append(current)
    current = input("Write something: ")
```

- Alternatywa: skonfigurowanie nieskończonej pętli `while` i użycie `break` - NIEZALECANE.

```
inputs = list()
while True:
    current = input("Write something: ")
    if current == "quit":
        break
    inputs.append(current)
```

- Poniższy kod prosi użytkownika o podanie danych wejściowych, dopóki nie wpisze "quit". Powtarzamy `input()` i musimy dodać `current()` do listy, zanim o to zapytamy:

```
1 inputs = list()
2 current = input("Write something: ")
3 while current != "quit":
4     inputs.append(current)
5     current = input("Write something: ")
```

- Alternatywa z operatorem morsa:

```
1 inputs = list()
2 while (current := input("Write something: ")) != "quit":
3     inputs.append(current)
```

```python
inputs = list()
current = input("Write something: ")
while current != "quit":
    inputs.append(current)
    current = input("Write something: ")
```

```python
inputs = list()
while True:
    current = input("Write something: ")
    if current == "quit":
        break
    inputs.append(current)
```

```python
inputs = list()
while (current := input("Write something: ")) != "quit":
    inputs.append(current)
```

- Użycie operatora morsa w pętli:

```
1 inputs = list()
2 while (current := input("Write something: ")) != "quit":
3     inputs.append(current)
```

- Użycie operatora morsa w instrukcji warunkowej:

```
1 a = [1, 2, 3, 4]
2 if (n := len(a)) > 3:
3     print(f"List is too long ({n} elements, expected <= 3)")
```

PEP 572 – Assignment Expressions – dokumentacja PEP 572 opisuje wszystkie szczegóły użycia operatora morsa, w tym niektóre uzasadnienia wprowadzenia ich do języka, a także kilka przykładów użycia operatora morsa.
https://www.python.org/dev/peps/pep-0572/

Odpowiednik do-while w Pythonie:

In[13]:

```
i = 1
while True:
    print(i)
    i += 1
    if i > 3:
        break
```

Out:

```
1
2
3
```

do-while w C:

In[14]:

```
int i = 1;
do{
    printf("%d\n", i);
    i = i + 1;
} while(i <= 3);
```

Out:

```
1
2
3
```

- Instrukcja `break` powoduje natychmiastowe wyjście z pętli (znajdziemy się poniżej całej instrukcji pętli), natomiast `continue` przeskakuje z powrotem na górę pętli (znajdziemy się tuż przed kolejnym elementem pobieranym w `while`).
- Część `else` w pętlach `while` zostanie wykonana raz, kiedy pętla się kończy — o ile kończy się normalnie, bez trafienia na instrukcję `break`. Instrukcja `break` powoduje natychmiastowe wyjście z pętli i pominięcie części `else` (o ile jest ona w ogóle obecna).
- **Jeśli można zamiast while należy używać prostych pętli for**. Prosta pętla for (na przykład `for x in seq:`) jest prawie zawsze łatwiejsza do zapisania w kodzie i szybsza do wykonania od pętli licznika opartego na `while`. Ponieważ Python wewnętrznie obsługuje indeksowanie dla prostego for, czasami pętla taka może być nawet dwa razy szybsza od odpowiadającej jej pętli while.

Dziękuje za uwagę

Kinga Węzka kinga.wezka@pw.edu.pl