Technical University of Lodz
Institute of Electronics

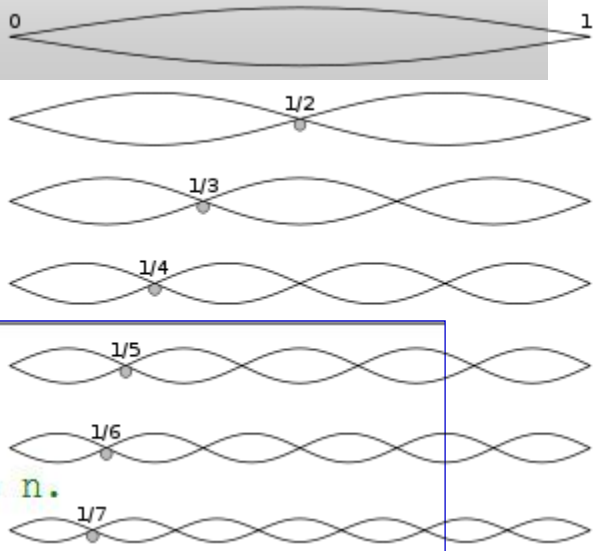
Algorytmy i struktury danych

4. Pętle i wyrażenia boolowskie

Łódź 2018

$$\sum_{n=1}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots$$

Suma szeregu harmonicznego



- Wpisz kod programu w oknie edycyjnym
- Zapisz kod w pliku **harmonic.py**
- Uruchom skrypt (`In[1]:` run harmonic.py)

```
1   # harmonic.py
2
3   def harmonic(n):
4       # Compute the sum of 1/k for every k from 1 to n.
5       total = 0.0
6       for k in range(1, n + 1):
7           total += 1.0 / k
8       return total
9
10  def main():
11      n = input('Enter a positive integer: ')
12      print "The sum of 1/k for k = 1 to %d is %.6f" % (n, harmonic(n))
13
14  main()
15
```

- Ten program wykorzystuje instrukcję **for**, funkcję **range()** oraz operator **+=**

Pętla **for**

- Pętla **for** jest używana do powtórzenia obliczeń w przypadkach, w których liczba powtórzeń jest znana z góry.

Składnia:

```
for <zmienna> in <sekwencja>:
    <ciało>
```

- Dla każdego elementu ciągu **<sekwencja>**, **<zmiennej>** zostaje przypisana wartość tego elementu, po czym wykonywane jest **<ciało>** pętli.

- W języku Python **sekwencją** jest uporządkowany zbiór elementów. Przykładem jest typ **list** (lista), którego elementy są wypisane w nawiasach kwadratowych, np. [12, -3, 5, 0.5].

```
In[1]: for x in [12, -3, 5, 0.5]:
In[2]:     print ( x, x**2)
```

- Co jest ciałem powyższej pętli?
- Ile razy jest obliczane ciało w ogólnym przypadku?

Funkcja **range**

- Jest to funkcja wbudowana, która generuje ciąg liczb całkowitych.

Składnia:

range(stop)	Rozpocznij od **0**. Dodaj kroki **1**. Zakończ przed **stop**.
range(start, stop)	Rozpocznij od **start**. Dodaj kroki 1. Zakończ przed **stop**.
range(start, stop, step)	Zacznij od **start**. Dodaj kroki **step**. Zakończ przed **stop**.

- Zmienne **start**, **stop** i **step** są typu całkowitego (**integer**).

```
In[1]: range(9)
In[2]: range(-2, 12)
In[3]: x = 5
In[4]: range(3, x+7)
In[5]: range(3, x-2)
In[6]: range(3, x-1)
In[7]: range(5, 2, -1)
```

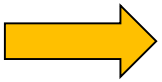
Pętla sumowania

- Skrypt ***harmonic.py*** zawiera zmienną sumowania **total**, która akumuluje wyniki kolejnych obliczeń w pętli.

Syntax:

```
<akumulator> = <wartość początkowa>
pętla:
        <akumulator> += <wartość dodawana>
```

- Instrukcja **x += v** jest *skróconym przypisaniem,* równoważnym **x = x + v**
- Pętle sumowania mogą akumulować wyniki obliczeń za pomocą innych operatorów, np. odejmowania, mnożenia, dzielenia (**x -= v**, **x *= v**, **x /= v**), nie tylko dodawania.



Nie używaj nazwy **sum** jako nazwy zmiennej akumulującej, ponieważ **sum()** jest wbudowaną funkcją języka Python.

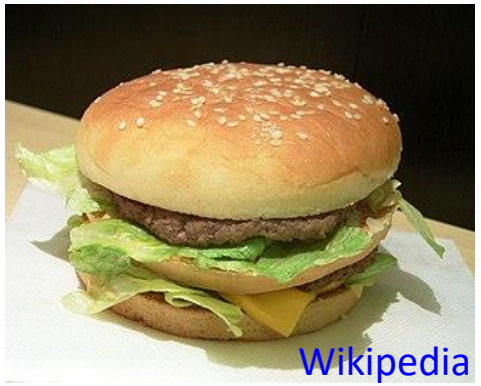
```
In[1]: x = range(6)
In[1]: print ( sum ( x ) )
```

- Jeśli liczba powtórzeń jest duża, czas wykonywania programu może stać się zbyt długi. Aby przerwać petlę, wciśnij **CTRL-D** albo zrestartuj *IPython*.

Body Mass Index … jeszcze raz

- Napisz skrypt, zapisz jako **bmi_evaluation.py** i uruchom go w środowisku ***IPython***.

$$\mathrm{BMI} = \frac{\mathrm{masa}_{\mathrm{kg}}}{\mathrm{wzrost}_{\mathrm{m}}^2}$$

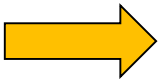



```
1   # BMI.py
2
3   YourName = str(input("Enter your name: "))
4   var = int(input("Enter your height in cm: "))
5   YourHeight = var / 100.0
6   YourMass = float(input("Enter your mass in kg: "))
7   BMI = YourMass / (YourHeight * YourHeight)
8   print("\n")
9   print("Hello " + YourName + "!")
10  if (BMI < 18.5):
11      print("Your Body Mass Index is %4.1f (possible underweight)." % (BMI))
12  elif (BMI >= 18.5) and (BMI < 25.0):
13      print("Your Body Mass Index is %4.1f (correct weight)." % (BMI))
14  else:
15      print("Your Body Mass Index is %4.1f (possible overweight)." % (BMI))
```

Wyrażenia boolowskie (logiczne)

- Innym typem danych języka Python jest typ **boolean**. Zmienne boolowskie wskazują tylko dwie możliwe wartości: **True** albo **False**.
- Poniższe **operatory relacji** zwracają wartości boolowskie:

x == y	Równy.
x != y	Nierówny.
x < y	Mniejszy.
x > y	Większy.
x <= y	Mniejszy lub równy.
x >= y	Większy lub równy.



Unikaj używania operatorów **==** oraz **!=** dla porównania wartości float!

```
In[1]: x = 3
In[2]: y = 2
In[3]: P = x == y     # P jest zmienną boolowską
In[4]: print (P)
```

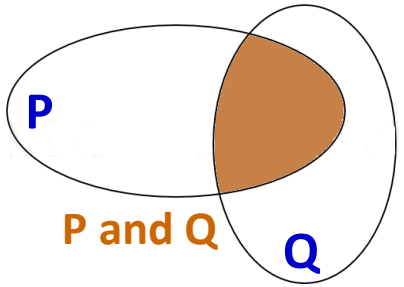
```
In [5]: a = 0.1+0.2
In [6]: a == 0.3
```

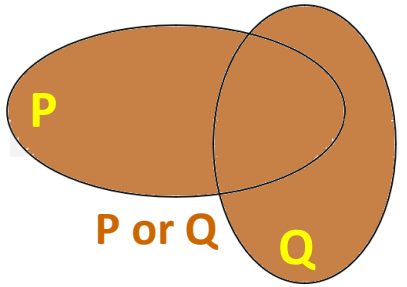
Wyrażenia boolowskie (logiczne)

- Wyrażenia boolowskie mogą być łączone z pomocą operatorów **boolowskich**

P and Q	**True** jeśli P i Q są **True**; jeśli nie: **False**.
P or Q	**True** jeśli P albo Q są **True**; jeśli nie: **False**.
not P	**True** jeśli P jest **False**; jeśli nie: **False**.

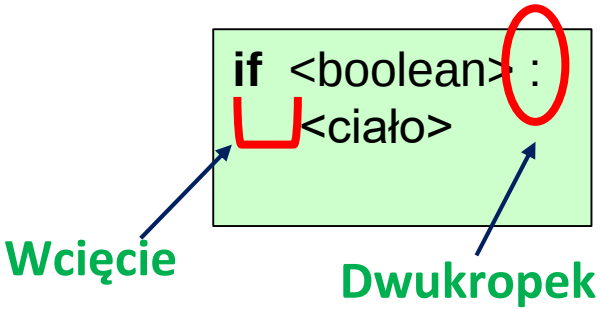
```
In[1]: x = 3
In[2]: y = 2
In[3]: P = not ((x == y) or (y == 2))
In[4]: print (P)

In[6]: P = True
In[7]: Q = False
In[8]: R = P and Q
In[9]: print R
```





Instrukcja warunkowa **if**

We wszystkich językach programowania występuje instrukcja **if** w jakiejś postaci. W języku **Python** ma ona zasadniczo 3 formy.




```
if <boolean> :
    <ciało1>
else:
    <ciało2>
```

```
if <boolean1> :
    <ciało1>
elif <boolean2> :
    <ciało2>
elif <boolean3> :
    <ciało3>
…
else :
    <ciałoN>
```

```
In[1]: YourGender = 'Female'
In[2]: if YourGender == 'Male' :
In[3]:     print ('Good day, Mister!')
In[4]: else:
In[5]:     print('Nice to see you, Madame!')
```

Pętla **while**

- Pętle **while** pozwalają na powtarzanie obliczeń w przypadkach, w których liczba powtórzeń nie jest znana z góry.

Składnia:

while <boolean> **:**
<ciało>

- Obliczana jest wartość wyrażenia **<boolean>**, a jeśli jest **True**, wykonywane są instrukcje **<ciała>**. Następnie wartość **<boolean>** jest znowu sprawdzana i jeśli nadal jest **True**, to **<ciało>** jest znowu obliczane. Cykl ten jest powtarzany, dopóki wyrażenie boolowskie przyjmie wartość **False**.

(a) Wypisz liczby całkowite od 1 do *n*

```
In[1]: n = 5
In[2]: i = 1
In[3]: while i < n+1:
In[4]:     print i
In[5]:     i += 1
```

(b) Wypisz liczby parzyste aż do *n*

```
In[1]: n = 10
In[2]: i = 1
In[3]: while i < n+2:
In[4]:     if i%2 == 0: print i
In[5]:     i += 1
```

Konwerter temperatury

- W poniższym programie **temp.py**, obliczenia w pętli **while** są powtarzane aż do chwili wprowadzenia niepoprawnej wartości temperatury.

```
1  # temp.py
2  # The entered temperature in Celsius
3  #    is converted to Fahrenheit degrees.
4  temp = input ( 'Enter a temperature in Celsius: ')
5  while temp<-273.15:
6      temp = input ('Impossible. Enter a valid temperature: ')
7  print "In Fahrenheit, that is", 9.0/5.0*temp+32.0
8
```

- Niepodobnie do pętli **for**, pętla **while** może być **nieskończona**. Naciśnij **CTRL-D** aby przerwać wykonywanie takiej pętli.

```
In[1]: i = 0
In[2]: while True:
In[3]:     print(i)
In[4]:     i += 1
```

Liczby pierwsze

- W poniższym programie **prime.py**, pętla **while** pozwala stwierdzić, czy wprowadzona liczba jest liczbą pierwszą, czy nie.

```
1  # prime.py
2
3  num = input ( 'Enter an integer number: ')
4  i = 2
5  # A prime is a number that divides by 1 and itself only.
6  while i<num and num%i != 0: # num%i equals to remainder
7      i += 1
8  if i==num:
9      print "The number", num, "is prime."
10 else:
11     print "The number", num, "is not prime."
12
```

Liczby pierwsze – analiza kodu

```
In[1]: i = 2
In[2]: while i<num and num%i != 0:
In[3]:     i += 1
```

Pętla zatrzymana

num = 9:

i	*i<num*	*num%i*	Boolean expression
2	T	1	T
3	T	0	F
-	-	-	-
-	-	-	-

Zatrzymanie: i = 3

i<num → „nie jest pierwsza”

num = 5:

i	*i<num*	*num%i*	Boolean expression
2	T	1	T
3	T	2	T
4	T	1	T
5	F	-	F

i = 5

i==num → „jest pierwsza”

Ćwiczenia

4.1. Wyjaśnij, dlaczego w wierszu 6 programu **harmonic.py** użyto **n+1** zamiast **n**. Jaka będzie wartość sumy szeregu dla bardzo dużych **n**?

4.2. Przypisz wartość **10** do zmiennej **n**, uruchom program **harmonic.py** i wprowadź wartość **3** jako liczbę **n**, o którą pyta program. Następnie wyświetl wartość **n** i objaśnij otrzymany rezultat. Wymień inne zmienne programu i objaśnij ich zasięg w programie.

4.3. Podaj wyrażenia **range**, do wygenerowania poniższych sekwencji
(a) 0, 1, 2, 3 (b) 3, 2, 1, 0 (c) 1, 3, 5, 7, 9, 11
(d) 10, 27, 44, 61, …, 197 (e) 100, 90, …, 10 (f) 2, 4, 6, 8, …, 200

4.4. Napisz program **myfactorial.py**, który zwraca iloczyn 1*2*3*…***n***. Zastosuj akumulator; nie używaj funkcji **factorial()** z modułu math.

Ćwiczenia

4.5. Napisz funkcję **mymax.py**, która zwraca największą z liczb *x, y* and *z*.

4.6. Napisz funkcję **median3.py**, która zwraca środkową liczbę spośród *x, y* and *z*.

4.7. Napisz funkcję **myabs.py** która zwraca wartość bezwzględną *x*.

4.8. Znaleź najmniejszą z liczb listy.

4.9. Znaleź największą z liczb listy.

4.10. Napisz kod pętli zanurzonej (nested loop). Jakie mogą być jej zastosowania?

4.11. Napisz kod pętli nieskończonej.

Podsumowanie

1) Petla **for** jest używana w przypadkach, w których wiadomo, ile razy należy powtórzyć obliczenia.
2) Dana typu **list** to uporządkowany ciąg elementów dowolnego typu, np. a_list = [10, False, ”Hi mate!”, 3.14159].
3) Dana typu **bolean** może mieć jedną z dwóch wartości: **True** or **False**.
4) Wartości boolowskie są zwracane przez **operatory relacji**.
5) Funkcja **range**() wytwarza listę liczb całkowitych.
6) Wyrażenie **if** pozwala na wykonanie różnych części programu, gdy spełnione są okreslone warunki.
7) Pętla **while** służy do programowania powtarzalnych obliczeń, przy czym liczba powtórzeń nie jest znana z góry.
8) W niektórych przypadkach pętle mogą być nieskończone.

Literatura

Brian Heinold, Introduction to Programming Using Python, Mount St. Mary's University, 2012 (http://faculty.msmary.edu/heinold/python.html).

Brad Dayley, Python Phrasebook: Essential Code and Commands, SAMS Publishing, 2007 (dostępne też tłumaczenie: B. Dayley, Python. Rozmówki, Helion, 2007).

Mark J. Johnson, A Concise Introduction to Programming in Python, CRC Press, 2012.