Politechnika Wrocławska

Wstęp do programowania

INP001126W

rok akademicki 2021/22

semestr zimowy

Wykład 5

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220

Politechnika Wrocławska

Plan prezentacji (1)

- Przykłady wykorzystania pętli:
  - Wczytywanie danych
  - Weryfikacja danych wejściowych
- Pętle zagnieżdżone

Na podstawie: T. Gaddis, *Python dla zupełnie początkujących*, Helion, 2019

Politechnika Wrocławska

Plan prezentacji (2)

- Importowanie modułów
- Wybrane funkcje z modułu **`random`**
- Wybrane funkcje z modułu **`math`**
- Tworzenie własnych modułów

Na podstawie: T. Gaddis, *Python dla zupełnie początkujących*, Helion, 2019

Politechnika Wrocławska

Wczytywanie danych z wartownikiem

- Do wczytywania danych o różnych długościach wykorzystywaliśmy dwie techniki:
  - na koniec każdej iteracji pytamy użytkownika, czy chce kontynuować,
  - na początku programu pytamy użytkownika, o długość danych.
- Dla długich list obie techniki są niewygodne:
  - w pierwszej trzeba wielokrotnie potwierdzać kontynuację,
  - w drugiej użytkownik musi znać liczbę elementów.

Wczytywanie danych z wartownikiem

- Rozwiązaniem może być wykorzystanie wartownika.
- Wartownik to specjalna wartość, która wskazuje koniec listy elementów.
- Przykładowo: w programie obliczającym średnią ocen – wprowadzenie wartości 0 (która nie jest poprawną oceną) może oznaczać koniec danych.

Politechnika Wrocławska

Wczytywanie danych z wartownikiem

- Wybór rozwiązania zależy od specyfiki problemu (znana długość danych? interakcja z użytkownikiem? możliwość wskazania wartości wartownika?)
- Wybór rozwiązania może także zależeć od preferencji programisty

Wczytywanie danych z wartownikiem

13_wartownik.py

File Edit Format Run Options Window Help

```
1  #Program pokazuje przykład użycia wartownika.
2  #
3  #Program oblicza średnią ocen.
4  #Przerywa działanie gdy użytkownik wprowadzi 0 jako ocenę.
5
6  def main():
7      print('Program oblicza średnią ocen.')
8      print('Aby zakończyć działanie programu podaj 0 jako ocenę.')
9      print('Średnia ocen wynosi',average_grade())
10
```

Wczytywanie danych z wartownikiem

13_wartownik.py

File Edit Format Run Options Window Help

```
# I: brak
# P: funkcja wczytuje dane od użytkownika obliczając jednocześnie
#    sumę wszystkich ocen oraz licząc ich liczbę;
#    pętla while jest kontynuowana dopóki oceną nie będzie 0;
#    zwracana wartość to iloraz sumy ocen i ich liczby
#    instrukcja if pozwala uniknąć dzielenia przez 0 jeśli
#    nie wczytano żadnych danych
# O: średnia ocen
def average_grade():
```

Wczytywanie danych z wartownikiem

13_wartownik.py

File Edit Format Run Options Window Help

```
19 def average_grade():
20     #inicjalizacja zmiennych
21     total   = 0.0 #suma wczytanych ocen
22     counter = 0   #liczba wczytanych ocen
23 
24     #odczytanie pierwszej oceny
25     grade   = get_grade()
26 
27     #pętla jest kontynuowana, jeśli ocena jest różna od zera
28     while grade != 0:
29         total   += grade    #wczytana ocena jest dodawana do sumy
30         counter += 1        #oraz zwiększany jest licznik ocen
31         print(counter,grade,total)
32         grade = get_grade() #wczytywana jest kolejna ocena
33 
34     #zakończenie obliczeń
35     #średnia obliczana jako suma ocen dzielona przez ich liczbę
36     #if zapobiega dzieleniu przez 0
37     if counter != 0:
38         # w tym przypadku obliczamy średnią i ją zwracamy
39         return total/counter
40     else:
41         # w tym przypadku zwracamy wartość None
42         return
```

Wczytywanie danych z wartownikiem

13_wartownik.py - D:\repozytoria\python-examples\pętle\13_wartownik.py (3.9.7)

File Edit Format Run Options Window Help

```
44 # I: brak
45 # P: wczytanie oceny (jako float)
46 # O: wczytana ocena
47 def get_grade():
48     return float(input('Podaj ocenę: '))
49
50 main()
51
52
```

Wczytywanie danych z wartownikiem - porównanie wersji

13_wartownik.py

File Edit Format Run Options Window Help

```
19 def average_grade():
20     #inicjalizacja zmiennych
21     total   = 0.0 #suma wczytanych ocen
22     counter = 0   #liczba wczytanych ocen
23 
24     #odczytanie pierwszej oceny
25     grade   = get_grade()
26 
27     #pętla jest kontynuowana, jeśli ocena jest różna od zera
28     while grade != 0:
29         total   += grade    #wczytana ocena jest dodawana do sumy
30         counter += 1        #oraz zwiększany jest licznik ocen
31         print(counter,grade,total)
32         grade = get_grade() #wczytywana jest kolejna ocena
33 
34     #zakończenie obliczeń
35     #średnia obliczana jako suma ocen dzielona przez ich liczbę
36     #if zapobiega dzieleniu przez 0
37     if counter != 0:
38         # w tym przypadku obliczamy średnią i ją zwracamy
39         return total/counter
40     else:
41         # w tym przypadku zwracamy wartość None
42         return
```

Wczytywanie danych z wartownikiem - porównanie wersji

13_wartownik_B.py

File Edit Format Run Options Window Help

```
19 def average_grade():
20     #inicjalizacja zmiennych
21     total   = 0.0 #suma wczytanych ocen
22     counter = 0   #liczba wczytanych ocen
23 
24     grade   = True #zmienna grad steruje przebiegiem pętli
25 
26     #pętla jest kontynuowana, jeśli ocena jest różna od zera
27     while grade:
28         grade = float(input('Podaj ocenę: '))
29         if grade:
30             total   += grade    #wczytana ocena jest dodawana do sumy
31             counter += 1        #oraz zwiększany jest licznik ocen
32         else:
33             print("OK. Kończę liczenie średniej")
34         print(counter,grade,total)
35 
36     #zakończenie obliczeń
37     #średnia obliczana jako suma ocen dzielona przez ich liczbę
38     #if zapobiega dzieleniu przez 0
39     if counter:
40         # w tym przypadku obliczamy średnią
41         total /= counter
42     else:
43         # w tym przypadku wynikiem jest NaN
44         total = float('NaN')
45 
46     #zwracamy obliczoną średnią
47     return total
```

Wczytywanie danych z wartownikiem

Najważniejsze różnice:

1. Dwukrotne vs. jednokrotne wywołanie wczytywania danych
2. Konieczność dwukrotnego sprawdzania warunku w drugiej wersji

Pozostałe różnice:

1. Różny zapis warunków w pętli **`while`** oraz instrukcjach **`if`**
2. Wykorzystanie wartości **`None/NaN`** do sygnalizacji wyjątkowej sytuacji
3. Różne umiejscowienie polecenia **`return`**
4. Wykorzystanie funkcji do obsługi wczytywania danych

Wczytywanie danych z wartownikiem

- Ten sam problem można rozwiązać na wiele różnych sposób
- Wybór konkretnego rozwiązania zależy od wielu czynników

Weryfikacja danych wejściowych

13_wartownik.py - D:\repozytoria\python-examples\pętle\13_wartownik.py (3.9.7)

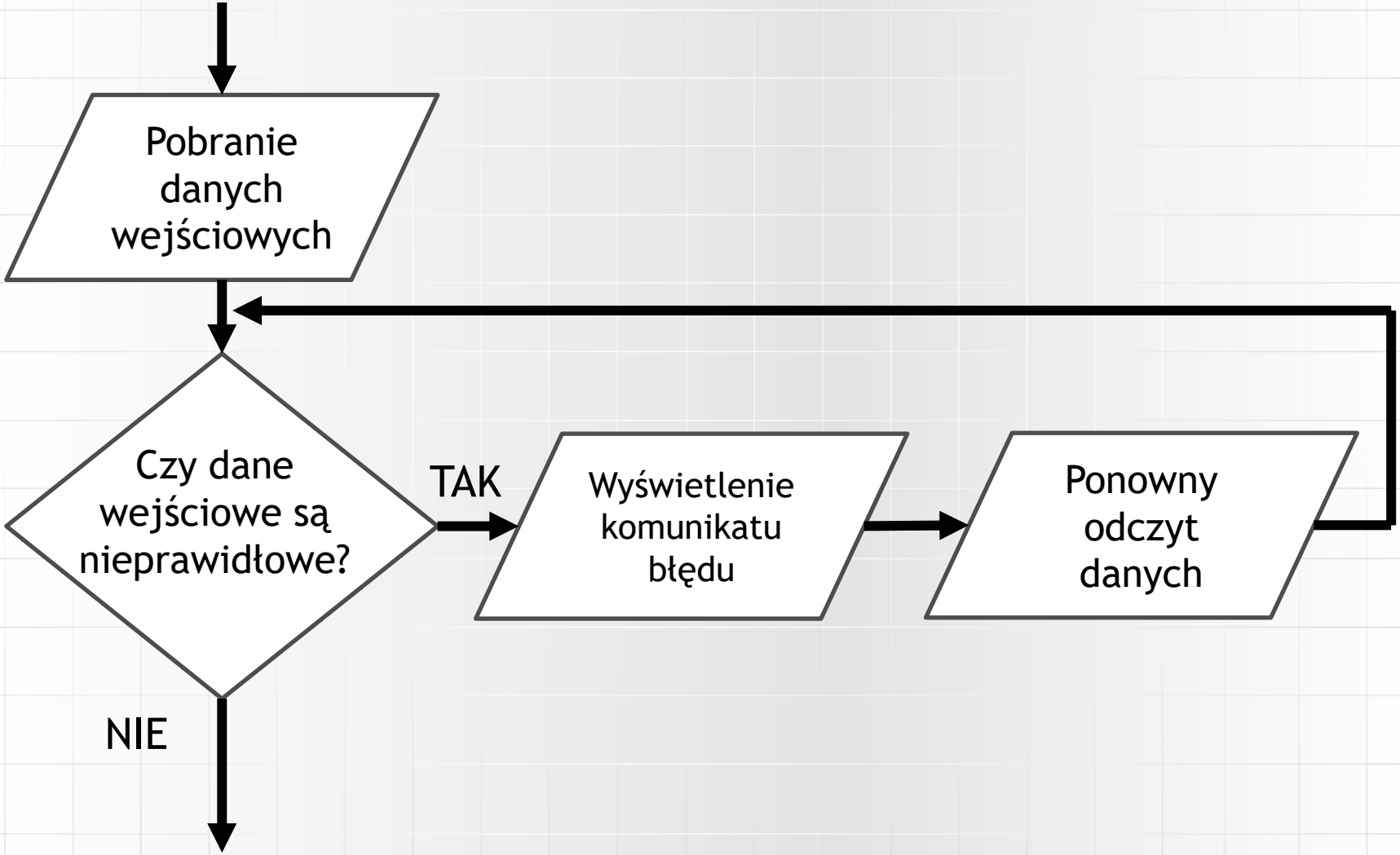
File Edit Format Run Options Window Help

```
44 # I: brak
45 # P: wczytanie oceny (jako float)
46 # O: wczytana ocena
47 def get_grade():
48     return float(input('Podaj ocenę: '))
49
50 main()
51
52
```

Weryfikacja danych wejściowych

- Proces sprawdzania poprawności danych, to weryfikacja danych wejściowych
- Dane najczęściej odczytuje się w pętli, wykonywanej dopóki dane są nieprawidłowe

Weryfikacja danych wejściowych



Weryfikacja danych wejściowych

14_weryfikacja_danych.py

File Edit Format Run Options Window Help

```
44 # I: brak
45 # P: wczytanie oceny (jako float)
46 #    jako prawidłowe dane przyjmowane są poprawne oceny lub wartość 0
47 # O: wczytana ocena
48 def get_grade():
49     grade = float(input('Podaj ocenę: '))
50     # sprawdzaj, czy wczytana liczba jest poprawną oceną lub zerem
51     # jeśli nie poproś o liczbę jeszcze raz
52     while not (is_grade(grade) or grade == 0.0):
53         print('Podana liczba nie jest oceną!')
54         grade = float(input('Podaj ocenę: '))
55     return grade
56
57 # I: liczba
58 # P: prawidłowa ocena jest jedną z wartości: 2, 3, 3.5, 4, 4.5, 5, 5.5;
59 # O: wartość logiczna - czy liczba jest oceną
60 def is_grade(number):
61     return number == 2.0 or \
62            number == 3.0 or number == 3.5 or \
63            number == 4.0 or number == 4.5 or \
64            number == 5.0 or number == 5.5
```

Politechnika Wrocławska

Pętle zagnieżdżone

- Wewnątrz pętli można także umieszczać pętle
- Taka pętla nazywana jest pętlą zagnieżdżoną

- Przykład pętli zagnieżdżonej – zegar
  Zegar odlicza w pętli godziny

Politechnika Wrocławska

Pętle zagnieżdżone

- Wewnątrz pętli można także umieszczać pętle
- Taka pętla nazywana jest pętlą zagnieżdżoną

- Przykład pętli zagnieżdżonej – zegar
  Zegar odlicza w pętli godziny, ale na każdą godzinę przypada pętla odliczająca minuty

Politechnika Wrocławska

Pętle zagnieżdżone

- Wewnątrz pętli można także umieszczać pętle
- Taka pętla nazywana jest pętlą zagnieżdżoną

- Przykład pętli zagnieżdżonej – zegar
  Zegar odlicza w pętli godziny, ale na każdą godzinę przypada pętla odliczająca minuty, a w każdej minucie jest pętla odliczająca sekundy.

Pętle zagnieżdżone

15_petle_zagniezdzone.py

File Edit Format Run Options Window Help

```
1  #Program ilustrujący działanie pętli for.
2  #Program wypisuje liczby od 0 do 59.
3  #Imituje odliczanie stopera.
4
5  def main():
6      for seconds in range(60):
7          print(seconds)
8
9  main()
10
```

Politechnika Wrocławska

Pętle zagnieżdżone

16_petle_zagniezdzone.py

File Edit Format Run Options Window Help

```
1  #Program ilustrujący działanie zagnieżdżonych
2  #pętli for.
3  #Imituje odliczanie stopera (przez godzinę).
4
5  def main():
6      for minutes in range(60):
7          for seconds in range(60):
8              print(minutes,seconds,sep=':')
9
10 main()
```

Politechnika Wrocławska

Pętle zagnieżdżone

17_petle_zagniezdzone.py

File Edit Format Run Options Window Help

```
1  #Program ilustrujący działanie zagnieżdżonych
2  #pętli for.
3  #Imituje odliczanie stopera (przez dobę).
4
5  def main():
6      for hours in range(24):
7          for minutes in range(60):
8              for seconds in range(60):
9                  print(hours,minutes,seconds,sep=':')
10
11 main()
```

Politechnika Wrocławska

Pętle zagnieżdżone

- Zmienna sterująca pętlą wewnętrzną może zależeć od wartości zmiennej kontrolującej pętlę zewnętrzną

Politechnika Wrocławska

Pętle zagnieżdżone

18_petle_zagniezdzone.py

File Edit Format Run Options Window Help

```
1  #Program "rysuje" na ekranie schodki o n stopniach.
2
3  def main():
4      print('Program wyświetla schodki o n stopniach')
5      n = int(input('Podaj n (dodatnią liczbę całkowitą): '))
6      stairs(n)
7
8  # I: funkcja pobiera liczbę całkowitą
9  # P: funkcja wypisuje n wierszy
10 #     w zerowym wierszu wypisuje zero spacji i znak '#'
11 #     w pierwszym wierszy jedną spację i znak '#'
12 #     w kolejnych wierszach zawsze o jedną spację więcej
13 #     w ten sposób powstają schodki
14 # O: funkcja nie zwraca wartości
15 def stairs(n):
16     #pętla po wierszach
17     for row in range(n):
18         #wypisanie spacji
19         #liczba spacji odpowiada - numerowi wiersza
20         for col in range(row):
21             print(' ', end='')
22         #wypisanie końcowego znaku '#'
23         print('#')
24
25 main()
```

Politechnika Wrocławska

Pętle zagnieżdżone

19_petle_zagniezdzone.py

File Edit Format Run Options Window Help

```
1  #Program "rysuje" na ekranie schodki o n stopniach.
2
3  def main():
4      print('Program wyświetla schodki o n stopniach')
5      n = int(input('Podaj n (dodatnią liczbę całkowitą): '))
6      stairs(n)
7
8  # I: funkcja pobiera liczbę całkowitą
9  # P: funkcja wypisuje n wierszy (schodków)
10 #    schodki wypisywane są w funkcji single_stair()
11 # O: funkcja nie zwraca wartości
12 def stairs(n):
13     for row in range(n):
14         single_stair(row)
15
16 # I: funkcja pobiera liczbę całkowitą (numer schodka)
17 # P: funkcja wypisuje n spacji oraz znak '#'
18 # O: funkcja nie zwraca wartości
19 def single_stair(n):
20     for x in range(n):
21         print(' ', end='')
22     print('#')
23
24
25 main()
```

Politechnika Wrocławska

Importowanie modułów

- Python jest dostarczany z biblioteką standardową, która zawiera gotowe funkcje
- Przykładowo: `print()`, `input()`, `range()`

- Część funkcji wbudowana jest w interpreter
- Wiele funkcji biblioteki standardowej umieszczono w modułach

Politechnika Wrocławska

Importowanie modułów

- Moduły pomagają w organizacji biblioteki standardowej
- Przykładowo: funkcje matematyczne przechowywane są razem w module, funkcje obsługujące pliki razem w innym module.
- Aby skorzystać w funkcji bibliotecznej zawartej w module, należy zaimportować moduł poleceniem **`import`**.

```
import nazwa_modułu
```

Politechnika Wrocławska

Importowanie modułów

- W przykładzie wykorzystamy moduł **`random`**, który zawiera definicje wielu funkcji pozwalających na generowanie liczb losowych (właściwie pseudolosowych).

Importowanie modułów

01_import_random_randint.py

File Edit Format Run Options Window Help

```
# Program pokazuje importowanie modułu (random)
# w celu wywołania funkcji zdefiniowanej w nim (randint).
# Program pokazuje wywołanie funkcji randint().
# Funkcja randint(a, b) zwraca losową liczbę całkowitą
# z przedziału od a do b (włącznie).

# Program symuluje pięć rzutów kostką sześcienną.

import random #import modułu

def main():
    #wykonaj pięć razy
    for i in range(5):
        #rzut kostką (losowa liczba od 1 do 6)
        number = random.randint(1,6)
        #wyświetl wynik
        print(number)

main()
```

Importowanie modułów

01_import_random_randint.py

File Edit Format Run Options Window Help

```
# Program pokazuje importowanie modułu (random)
# w celu wywołania funkcji zdefiniowanej w nim (randint).
# Program pokazuje wywołanie funkcji randint().
# Funkcja randint(a, b) zwraca losową liczbę całkowitą
# z przedziału od a do b (włącznie).

# Program symuluje pięć rzutów kostką sześcienną.

import random #import modułu

def main():
    #wykonaj pięć razy
    for i in range(5):
        #rzut kostką (losowa liczba od 1 do 6)
        number = random.randint(1,6)
        #wyświetl wynik
        print(number)

main()
```

Wybrane funkcje z modułu `random`

- W module **`random`** dostępna jest funkcja **`randint()`**, która zwraca losową liczbę całkowitą z podanego przedziału.

Wybrane funkcje z modułu `random`

01_import_random_randint.py

File Edit Format Run Options Window Help

```
# Program pokazuje importowanie modułu (random)
# w celu wywołania funkcji zdefiniowanej w nim (randint).
# Program pokazuje wywołanie funkcji randint().
# Funkcja randint(a, b) zwraca losową liczbę całkowitą
# z przedziału od a do b (włącznie).

# Program symuluje pięć rzutów kostką sześcienną.

import random #import modułu

def main():
    #wykonaj pięć razy
    for i in range(5):
        #rzut kostką (losowa liczba od 1 do 6)
        number = random.randint(1,6)
        #wyświetl wynik
        print(number)

main()
```

Wybrane funkcje z modułu `random`

02_randint.py

File Edit Format Run Options Window Help

```
# Program symuluje serię rzutów monetą.
# Program wykorzystuje funkcję randint()
# z modułu random.

import random #import modułu

HEAD = 1    #stała oznaczająca awers (orzeł)
TAIL = 2    #stała oznaczająca rewers (reszka)
TOSSES = 10 #stała określająca liczbę rzutów

def main():
    # w każdym rzucie
    for toss in range(TOSSES):
        # wylosuj liczbę (1 lub 2)
        # 1 oznacza awers (orzeł)
        if random.randint(HEAD, TAIL) == HEAD:
            print('orzeł')
        # w przeciwnym wypadku to rewers (reszka)
        else:
            print('reszka')

main()
```

Politechnika Wrocławska

Wybrane funkcje z modułu `random`

Inne funkcje dostępne w module **`random`**:

- **`randrange()`**
- **`random()`**
- **`uniform()`**

Wybrane funkcje z modułu `random`

03_random.py

File Edit Format Run Options Window Help

```
# Program pokazuje różne funkcje z modułu random.

import random

def main():
    #randrange - zwraca losową liczbę z listy
    print(random.randrange(6))          #losowa wartość od 0 do 5
    print(random.randrange(7,10))       #losowa wartość od 7 do 9
    print(random.randrange(0,101,10)) #losowa wartość z listy od 0 co 10 do 100
    #random - zwraca losową liczbę z przedziału [0.0, 1.0)
    print(random.random())
    #uniform - zwraca losową liczbę z przedziału [a, b]
    print(random.uniform(3.0,5.0))     #losowa wartość z przedziału [3, 5]

main()
```

Wybrane funkcje z modułu `random`

- Liczby pseudolosowe generowane są jako wyrazy deterministycznego ciągu bazującego na pewnej informacji początkowej (nazywanej ziarnem lub zalążkiem).
- Wartość zalążka można zadać funkcją **`seed()`**.
- Kontrolowanie zalążka generatora liczb pseudolowych pozwala odtworzyć działanie programu wykorzystującego liczby pseudolosowe (np. na potrzeby debuggowania kodu programu)

Wybrane funkcje z modułu `random`

04_random.py

File Edit Format Run Options Window Help

```
# Program pokazuje działanie funkcji seed().
# Program symuluje pięć rzutów kostką sześcienną.
# Wartości uzyskiwane z każdego uruchomienia
# są takie same - dzięki ustawieniu ziarna generatora.

import random

def main():
    # ustawienie ziarna generatora
    random.seed(10)
    for i in range(5):
        print(random.randint(1,6))

main()
```

Wybrane funkcje z modułu `math`

- Moduł **`math`** zawiera funkcje matematyczne
- Przykładowo, w module **`math`** dostępna jest funkcja **`sqrt()`**, obliczająca pierwiatek kwadratowy.
- Przykładowo, w module **`math`** dostępna jest zmienna **`pi`**, przechowująca przybliżenie liczby π.

Wybrane funkcje z modułu `math`

05_import_math.py

File Edit Format Run Options Window Help

```
# Program pokazuje działanie funkcji sqrt()
# z modułu math.

#importowanie modułu math
import math

def main():
    #wczytanie liczby rzeczywistej
    number = float(input('Podaj liczbę: '))
    #obliczenie pierwiastka
    #funkcja sqrt() z biblioteki math
    square_root = math.sqrt(number)
    #wyświetlenie wyników
    print('Pierwiastek kwadratowy liczby',
          number,'wynosi',square_root)

main()
```

Wybrane funkcje z modułu `math`

06_math_pi.py

File Edit Format Run Options Window Help

```
# Program oblicza pole koła o podanym promieniu.
# Program wykorzystuje wielkość pi z modułu math.

import math

def main():
    #wczytanie liczby rzeczywistej
    radius = float(input('Podaj promień koła: '))
    area   = math.pi * radius ** 2
    #wyświetlenie wyników
    print('Pole koła o promieniu',
          radius,'wynosi',area)

main()
```

Politechnika Wrocławska

Wybrane funkcje z modułu `math`

Funkcja	Opis
`acos(x)`	arcus cosinus x (w radianach)
`asin(x)`	arcus sinus x (w radianach)
`atan(x)`	arcus tan x (w radianach)
`ceil(x)`	zaokrąglenie w górę (sufit)
`cos(x)`	cosinus dla x (wyrażonego w radianach)
`exp(x)`	e^x
`floor(x)`	zaokrąglenie w dół (podłoga)
`log(x)`	logarytm naturalny x
`log10(x)`	logarytm dziesiętny x
`sin(x)`	sinus dla x (wyrażonego w radianach)
`sqrt(x)`	pierwiastek kwadratowy z x
`tan(x)`	tangens dla x (wyrażonego w radianach)

Politechnika Wrocławska

Tworzenie własnych modułów

- Złożone programy powinny być podzielone na funkcje
- Kod programu zawierającego wiele funkcji można podzielić na moduły
- Przykładowy program oblicza pola i obwody kół i prostokątów

Politechnika Wrocławska

Tworzenie własnych modułów

- Funkcje obsługujące koło zebrano w module (plik circle.py).
- Funkcje obsługujące prostokąt zebrano w module (plik rectangle.py)
- Program importuje te moduły

Politechnika Wrocławska

Tworzenie własnych modułów

circle.py

File Edit Format Run Options Window Help

```
# Moduł circle.py
# Zawiera definicje funkcji służących do obliczania:
# pola i obwodu koła oraz do wczytywania promienia.

# import modułu math - wykorzystywana wartość math.pi
import math

# Funkcja oblicza pole koła.
# I: promień
# P: oblicza pole z zależności: pole = pi*r^2
# O: obliczona wartość pola
def area(radius):
    return math.pi * radius ** 2

# Funkcja oblicza obwód koła.
# I: promień
# P: oblicza obwód z zależności: obwód = 2*pi*r
# O: obliczona wartość obwodu
def circumference(radius):
    return 2 * math.pi * radius

# Funkcja pobiera promień koła od użytkownika.
# I: brak
# P: wczytanie liczby rzeczywistej podanej przez użytk
# O: promień
def get_radius():
    return float(input('Podaj promień: '))
```

Tworzenie własnych modułów

rectangle.py

File Edit Format Run Options Window Help

```
# Moduł rectangle.py
# Zawiera definicje funkcji służących do obliczania:
# pola i obwodu prostokąta oraz
# do wczytywania jego wymiarów.

# Funkcja oblicza pole prostokąta.
# I: długość i szerkość
# P: oblicza pole z zależności: pole = długość*szerkość
# O: obliczona wartość pola
def area(length, width):
    return length * width

# Funkcja oblicza obwód prostokąta.
# I: długość i szerkość
# P: oblicza obwód z zależności: obwód = 2*(długość+szerkość)
# O: obliczona wartość obwodu
def perimeter(length, width):
    return 2 * (length + width)

# Funkcja pobiera wymiary prostokąta od użytkownika.
# I: brak
# P: wczytanie liczby rzeczywistej podanej przez użytkownika
# O: (długość, szerkość)
def get_dimensions():
    length = float(input('Podaj długość prostokąta: '))
    width  = float(input('Podaj szerokość prostokąta: '))
    return (length, width)
```

Tworzenie własnych modułów

07_geometry.py

File Edit Format Run Options Window Help

```
# Program pokazuje importowanie własnych modułów.
# Program wykorzystuje pliki circle.py oraz rectangle.py.
#
# Plik circle.py zawiera definicje funkcji:
# - area(radius),
# - circumference(radius),
# - get_radius().
#
# Plik rectangle.py zawiera definicje funkcji:
# - area(length, width),
# - perimeter(length, width),
# - get_dimensions().

#importowanie własnych modułów
import circle
import rectangle
```

Tworzenie własnych modułów

07_geometry.py

File Edit Format Run Options Window Help

```
#definicje stałych do obsługi menu
AREA_CIRCLE_CHOICE          = 1
CIRCUMFERENCE_CHOICE        = 2
AREA_RECTANGLE_CHOICE       = 3
PERIMETER_RECTANGLE_CHOICE  = 4
QUIT_CHOICE                 = 5

def main():
    print('Program geometria pozwala obliczyć pole i obwód koła i prostokąta.')

    # W pętli while pobierana jest od użytkownika liczba całkowita
    # odpowiadająca wybranej czynności (od 1 do 5), a następnie
    # w bloku if-elif-else odczytywane są dane i wykonywane obliczenia.
    choice = 0 #inicjalizacja zmiennej choice
    while choice != QUIT_CHOICE:
        display_menu() #wyświetlenie menu
        #pobranie wartości od użytkownika
        choice = int(input('Wybierz opcję: '))

        #wybór czynności
        if choice == AREA_CIRCLE_CHOICE:            #pole koła
            radius = circle.get_radius()
            print('Pole koła wynosi',circle.area(radius))
        elif choice == CIRCUMFERENCE_CHOICE:        #obwód koła
            radius = circle.get_radius()
            print('Obwód koła wynosi',circle.circumference(radius))
        elif choice == AREA_RECTANGLE_CHOICE:       #pole prosokąta
            (length, width) = rectangle.get_dimensions()
            print('Pole prostokąta wynosi', rectangle.area(length, width))
        elif choice == PERIMETER_RECTANGLE_CHOICE:  #obwód prostokąta
            (length, width) = rectangle.get_dimensions()
            print('Obwód prostokąta wynosi', rectangle.perimeter(length, width))
        elif choice == QUIT_CHOICE:                 #koniec programu
            print('Zakończenie działania programu.')
        else:                                       #wartość spoza listy
            print('Błąd: nieprawidłowa opcja.')
```

Wykorzystanie funkcji z modułów **`circle`** oraz **`rectangle`**

Tworzenie własnych modułów

07_geometry.py

File Edit Format Run Options Window Help

```
55 def display_menu():
56     print("""
57 Co chcesz zrobić?
58 1. Policz pole koła.
59 2. Policz obwód koła.
60 3. Policz pole prostokąta.
61 4. Policz obwód prostokąta.
62 5. Zakończ pracę.""")
63
64 main()
65
```

Politechnika Wrocławska

Tworzenie własnych modułów

- Przykładowy program, wykorzystuje pętlę **`while`** oraz instrukcję **`if-elif-else`** do obsługi menu

Politechnika Wrocławska

Absolutne minimum (1)

- Trzy sposoby wczytywania danych
- Weryfikacja danych wejściowych w pętli
- Pętle zagnieżdżone

Politechnika Wrocławska

Absolutne minimum (2)

- Importowanie modułów biblioteki standardowej
- Znajomość wybranych funkcji z modułów **`random`** oraz **`math`**