Politechnika Wrocławska

Podstawy programowania W11OPA-SI0072G
Wstęp do programowania W11FTE-SI0141WL
Wstęp do programowania W11IKW-SI0080WL

rok akademicki 2023/24
semestr letni

Wykład 7

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220

Politechnika Wrocławska

Plan prezentacji (1)

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

T. Gaddis, *Python dla zupełnie początkujących*, Helion, 2019

Politechnika Wrocławska

Plan prezentacji (2)

- Funkcje rekurencyjne
  - Co to jest rekurencja?
  - Warunek stopu
  - Przykłady algorytmów rekurencyjnych

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).

Politechnika Wrocławska

Importowanie modułów

01_import_random_randint.py

```python
# 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()
```

Politechnika Wrocławska

Importowanie modułów

01_import_random_randint.py

```python
# 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()
```

Politechnika Wrocławska

Wybrane funkcje z modułu `random`

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

Politechnika Wrocławska

Importowanie modułów

01_import_random_randint.py

```
1  # Program pokazuje importowanie modułu (random)
2  # w celu wywołania funkcji zdefiniowanej w nim (randint).
3  # Program pokazuje wywołanie funkcji randint().
4  # Funkcja randint(a, b) zwraca losową liczbę całkowitą
5  # z przedziału od a do b (włącznie).
6
7  # Program symuluje pięć rzutów kostką sześcienną.
8
9  import random #import modułu
10
11 def main():
12     #wykonaj pięć razy
13     for i in range(5):
14         #rzut kostką (losowa liczba od 1 do 6)
15         number = random.randint(1,6)
16         #wyswietl wynik
17         print(number)
18
19 main()
```

Politechnika Wrocławska

Wybrane funkcje z modułu `random`

01_import_random_randint.py | 02_randint.py

```python
# 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()`**

Politechnika Wrocławska

Wybrane funkcje z modułu `random`

01_import_random_randint.py | 02_randint.py | 03_random.py

```python
# 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).

Politechnika Wrocławska

Wybrane funkcje z modułu `random`

01_import_random_randint.py | 02_randint.py | 03_random.py | 04_random.py

```
1  # Program pokazuje działanie funkcji seed().
2  # Program symuluje pięć rzutów kostką sześcienną.
3  # Wartości uzyskiwane z każdego uruchomienia
4  # są takie same - dzięki ustawieniu ziarna generatora.
5
6  import random
7
8  def main():
9      # ustawienie ziarna generatora
10     random.seed(10)
11     for i in range(5):
12         print(random.randint(1,6))
13
14 main()
```

Politechnika Wrocławska

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.

Politechnika Wrocławska

Wybrane funkcje z modułu `math`

05_import_math.py

```python
# 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()
```

Politechnika Wrocławska

Wybrane funkcje z modułu `math`

05_import_math.py | 06_math_pi.py

```
1  # Program oblicza pole koła o podanym promieniu.
2  # Program wykorzystuje wielkość pi z modułu math.
3
4  import math
5
6  def main():
7      #wczytanie liczby rzeczywistej
8      radius = float(input('Podaj promień koła: '))
9      area   = math.pi * radius ** 2
10     #wyświetlenie wyników
11     print('Pole koła o promieniu',
12           radius,'wynosi',area)
```

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 | rectangle.py | 07_geometry.py

```python
# 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 użytkownika
# O: promień
def get_radius():
    return float(input('Podaj promień: '))
```

Politechnika Wrocławska

Tworzenie własnych modułów

circle.py | rectangle.py | 07_geometry.py

```python
# 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)
```

Politechnika Wrocławska

Tworzenie własnych modułów

circle.py | rectangle.py | 07_geometry.py

```
1  # Program pokazuje importowanie własnych modułów.
2  # Program wykorzystuje pliki circle.py oraz rectangle.py.
3  #
4  # Plik circle.py zawiera definicje funkcji:
5  # - area(radius),
6  # - circumference(radius),
7  # - get_radius().
8  #
9  # Plik rectangle.py zawiera definicje funkcji:
10 # - area(length, width),
11 # - perimeter(length, width),
12 # - get_dimensions().
13
14 #importowanie własnych modułów
15 import circle
16 import rectangle
17
```

Politechnika Wrocławska

Tworzenie własnych modułów

circle.py | rectangle.py | 07_geometry.py

```
14  #importowanie własnych modułów
15  import circle
16  import rectangle
17
18  #definicje stałych do obsługi menu
19  AREA_CIRCLE_CHOICE         = 1
20  CIRCUMFERENCE_CHOICE       = 2
21  AREA_RECTANGLE_CHOICE      = 3
22  PERIMETER_RECTANGLE_CHOICE = 4
23  QUIT_CHOICE                = 5
24
25  def main():
26      print('Program geometria pozwala obliczyć pole i obwód koła i prostokąta.')
27
28      # W pętli while pobierana jest od użytkownika liczba całkowita
29      # odpowiadająca wybranej czynności (od 1 do 5), a następnie
30      # w bloku if-elif-else odczytywane są dane i wykonywane obliczenia.
31      choice = 0 #inicjalizacja zmiennej choice
32      while choice != QUIT_CHOICE:
33          display_menu() #wyświetlenie menu
34          #pobranie wartości od użytkownika
35          choice = int(input('Wybierz opcję: '))
36
37          #wybór czynności
```

Politechnika Wrocławska

Tworzenie własnych modułów

circle.py | rectangle.py | 07_geometry.py

```
32      while choice != QUIT_CHOICE:
33          display_menu() #wyświetlenie menu
34          #pobranie wartości od użytkownika
35          choice = int(input('Wybierz opcję: '))
36
37          #wybór czynności
38          if choice == AREA_CIRCLE_CHOICE:            #pole koła
39              radius = circle.get_radius()
40              print('Pole koła wynosi',circle.area(radius))
41          elif choice == CIRCUMFERENCE_CHOICE:        #obwód koła
42              radius = circle.get_radius()
43              print('Obwód koła wynosi',circle.circumference(radius))
44          elif choice == AREA_RECTANGLE_CHOICE:       #pole prosokąta
45              (length, width) = rectangle.get_dimensions()
46              print('Pole prostokąta wynosi', rectangle.area(length, width))
47          elif choice == PERIMETER_RECTANGLE_CHOICE:  #obwód prostokąta
48              (length, width) = rectangle.get_dimensions()
49              print('Obwód prostokąta wynosi', rectangle.perimeter(length, width))
50          elif choice == QUIT_CHOICE:                 #koniec programu
51              print('Zakończenie działania programu.')
52          else:                                       #wartość spoza listy
53              print('Błąd: nieprawidłowa opcja.')
```

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

Politechnika Wrocławska

Tworzenie własnych modułów

circle.py | rectangle.py | 07_geometry.py

```
54
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

Funkcje rekurencyjne

- Rekurencją nazywamy odwoływanie się funkcji do samej siebie
- Rekurencję można wykorzystać do rozwiązywania problemów, które można podzielić na mniejsze wersje tego samego problemu

Politechnika Wrocławska

Funkcje rekurencyjne

recursion_endless.py | recursion_01.py

```
1  #Program pokazuje przykładową funkcję rekurencyjną.
2
3  #Jest to przykład nieskończonej rekurencji.
4
5  def main():
6      message()
7
8  #Funkcja message() wywołuje samą siebie.
9  def message():
10     print('To jest funkcja rekurencyjna')
11     message()
12
13 main()
14
```

Funkcje rekurencyjne

- Istotnym zagadnieniem przy wykorzystaniu rekurencji jest warunek stopu

Politechnika Wrocławska

Funkcje rekurencyjne

recursion_endless.py | recursion_01.py

```python
#Program pokazuje przykładową funkcję rekurencyjną.

def main():
    message(5)

#I :liczba całkowita określająca liczbę
#   wywołań rekurencyjnych
#P :jeśli liczba wywołań jest dodatnia
#       wyświetl informację
#       wywołaj rekurencyjnie funkcję message()
#           z liczbą wywołań mniejszą o jeden
#O :brak
def message(times):
    if times > 0:
        print('To jest funkcja rekurencyjna')
        message(times - 1)

main()
```

Politechnika Wrocławska

Funkcje rekurencyjne

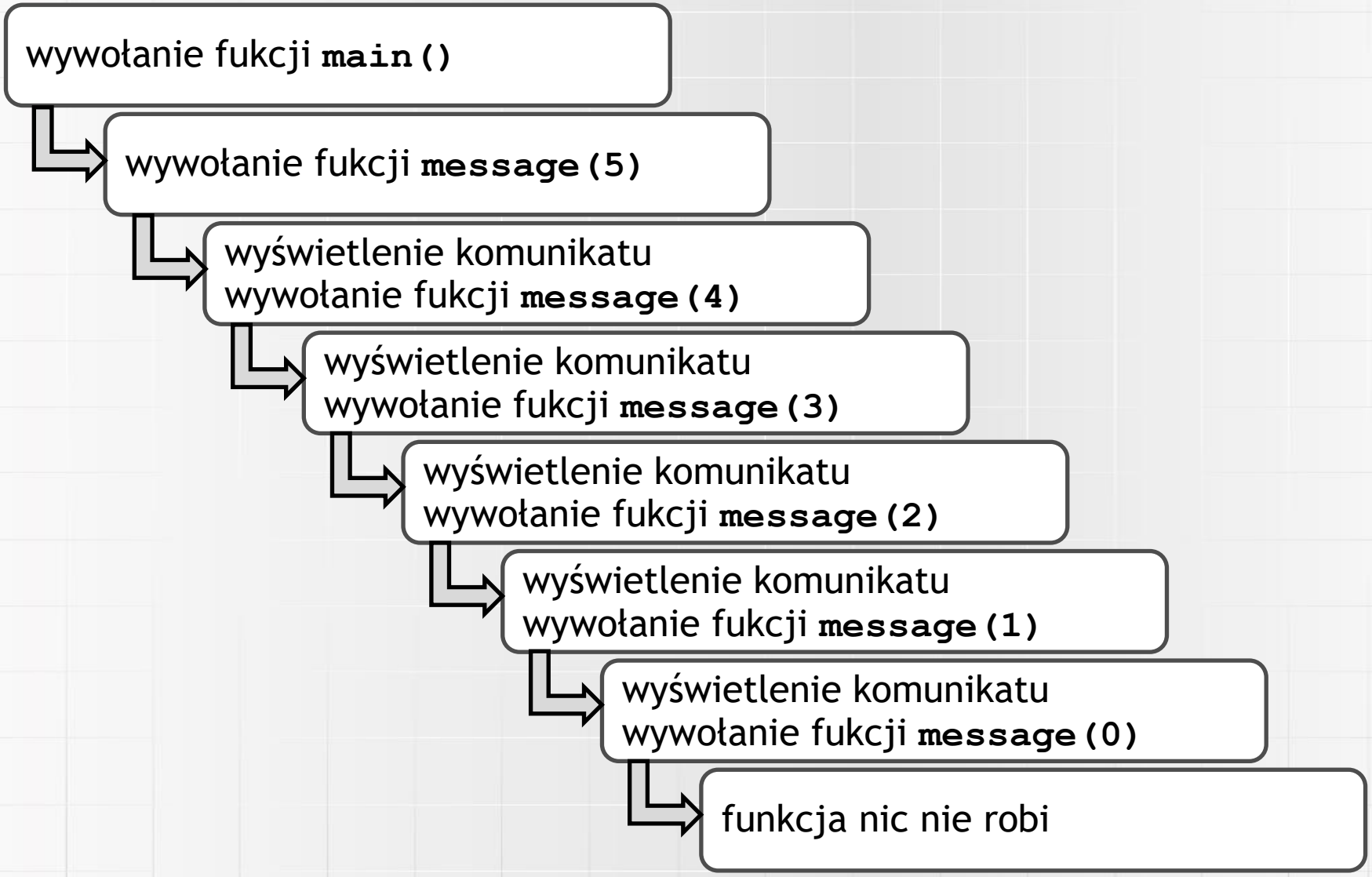
recursion_endless.py | recursion_01.py

```
1  #Program pokazuje przykładową funkcję rekurencyjną.
2
3  def main():
4      message(5)
5
6  #I :liczba całkowita określająca liczbę
7  #   wywołań rekurencyjnych
8  #P :jeśli liczba wywołań je
9  #       wyświetl informację
10 #       wywołaj rekurencyjn
11 #           z liczbą wywoła
12 #O :brak
13 def message(times):
14     if times > 0:
15         print('To jest funkcja rekurencyjna')
16         message(times - 1)
17
18 main()
19
```

```
To jest funkcja rekurencyjna
To jest funkcja rekurencyjna
To jest funkcja rekurencyjna
To jest funkcja rekurencyjna
To jest funkcja rekurencyjna
```

Politechnika Wrocławska

Funkcje rekurencyjne



Politechnika Wrocławska

Funkcje rekurencyjne

recursion_endless.py | recursion_01.py | recursion_02.py

```python
#Program pokazuje przykładową funkcję rekurencyjną.

def main():
    message(5)

#I :liczba całkowita określająca liczbę
#   wywołań rekurencyjnych
#P :jeśli liczba wywołań jest dodatnia
#       wyświetl informację
#       wywołaj rekurencyjnie funkcję message()
#           z liczbę wywołań mniejszą o jeden
#O :brak
def message(times):
    if times > 0:
        print('Wartość parametru times', times)
        message(times - 1)

main()
```

Politechnika Wrocławska

Funkcje rekurencyjne

recursion_endless.py | recursion_01.py | recursion_02.py

```
1  #Program pokazuje przykładową funkcję rekurencyjną.
2
3  def main():
4      message(5)
5
6  #I :liczba całkowita określają
7  #   wywołań rekurencyjnych
8  #P :jeśli liczba wywołań jest
9  #       wyświetl informację
10 #       wywołaj rekurencyjnie
11 #           z liczbę wywołań m
12 #O :brak
13 def message(times):
14     if times > 0:
15         print('Wartość parametru times', times)
16         message(times - 1)
17
18 main()
```

```
rekurencyjne')
Wartość parametru times 5
Wartość parametru times 4
Wartość parametru times 3
Wartość parametru times 2
Wartość parametru times 1
```

Politechnika Wrocławska

Funkcje rekurencyjne

recursion_endless.py | recursion_01.py | recursion_02.py | recursion_03.py

```
1  #Program pokazuje przykładową funkcję rekurencyjną.
2
3  def main():
4      message(5)
5
6  #I :liczba całkowita określająca liczbę wywołań
7  #   rekurencyjnych
8  #P :jeśli liczba wywołań jest dodatnia
9  #       wywołaj rekurencyjnie funkcję message()
10 #           z liczbę wywołań mniejszą o jeden
11 #       wyświetl informację
12 #O :brak
13 def message(times):
14     if times > 0:
15         #zmieniona kolejność instrukcji
16         message(times - 1)
17         print('Wartość parametru times', times)
18
19 main()
20
```

Politechnika Wrocławska

Funkcje rekurencyjne

recursion_endless.py | recursion_01.py | recursion_02.py | recursion_03.py

```
1  #Program pokazuje przykładową funkcję rekurencyjną.
2
3  def main():
4      message(5)
5
6  #I :liczba całkowita określająca liczbe wywołań
7  #   rekurencyjnych
8  #P :jeśli liczba wywołań jest d
9  #       wywołaj rekurencyjnie f
10 #           z liczbę wywołań mn
11 #       wyświetl informację
12 #O :brak
13 def message(times):
14     if times > 0:
15         #zmieniona kolejność instrukcji
16         message(times - 1)
17         print('Wartość parametru times', times)
18
19 main()
20
```

```
rekurencyjne')
Wartość parametru times 1
Wartość parametru times 2
Wartość parametru times 3
Wartość parametru times 4
Wartość parametru times 5
```

Politechnika Wrocławska

Funkcje rekurencyjne

Silnia

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$

$$n! = \begin{cases} 1 & \text{dla } n = 0, \\ n \cdot (n-1)! & \text{dla } n > 0. \end{cases}$$

Politechnika Wrocławska

Funkcje rekurencyjne

Silnia

recursion_endless.py | recursion_01.py | recursion_02.py | recursion_03.py | factorial.py

```
1  #Program demonstruje działanie funkcji factorial
2  #obliczającej wartość silni rekurencyjnie
3
4  def main():
5      print('5! =',factorial(5))
6
7  #I :liczba całkowita n
8  #P :oblicz silnię liczby n jako iloczyn
9  #   n * silnia (n-1) (jeśli n jest różne od zera)
10 #   dla zera - warunek stopu
11 #   0! = 1
12 #O :silnia liczby n
13
14 def factorial(n):
15     if n == 0:
16         return 1
17     else:
18         return n*factorial(n-1)
19
20 main()
```

Politechnika Wrocławska

Funkcje rekurencyjne

Silnia

recursion_endless.py | recursion_01.py | recursion_02.py | recursion_03.py | factorial.py | factorial_02.py

```
1  #Program demonstruje działanie funkcji factorial
2  #obliczającej wartość silni rekurencyjnie
3
4  def main():
5      print('5! =',factorial(5))
6
7  def factorial(n, level = 0):
8      print(f"level = {level}", level*"   ", f"factorial({n})")
9      if n == 0:
10         print(f"level = {level}", level*"   ", 1)
11         return 1
12     else:
13         x = n*factorial(n-1, level + 1)
14         print(f"level = {level}", level*"   ", x)
15         return x
16
17 main()
```

Politechnika Wrocławska

Funkcje rekurencyjne

Silnia

recursion_endless.py | recursion_01.py | recursion_02.py | recursion_03.py | factorial.py | factorial_02.py

```
1  #Program demonstruje działanie funkcji factorial
2  #obliczającej wartość silni
3
4  def main():
5      print('5! =',factorial(
6
7  def factorial(n, level = 0)
8      print(f"level = {level}
9      if n == 0:
10         print(f"level = {le
11         return 1
12     else:
13         x = n*factorial(n-1
14         print(f"level = {le
15         return x
16
17 main()
```

```
level = 0  factorial(5)
level = 1     factorial(4)
level = 2        factorial(3)
level = 3           factorial(2)
level = 4              factorial(1)
level = 5                 factorial(0)
level = 5                 1
level = 4              1
level = 3           2
level = 2        6
level = 1     24
level = 0  120
5! = 120
```

Politechnika Wrocławska

Algorytmy rekurencyjne

- Algorytm Euklidesa wyznaczania największego wspólnego dzielnika
- Potęgowanie rekurencyjne
- Wyznaczania elementów ciągu Fibonacciego

Politechnika Wrocławska

Algorytm Euklidesa

zakładając, że $m \leq n$

$$\mathrm{NWD}(m,n) = \begin{cases} n, & \text{dla } m = 0, \\ \mathrm{NWD}(n \bmod m, m), & \text{dla } m > 0. \end{cases}$$

Politechnika Wrocławska

Obliczanie potęgi

$$x^m = \begin{cases} x, & \text{dla } m = 1 \\ \left(x^{m/2}\right)^2, & m - \text{parzyste} \\ \left(x^{(m-1)/2}\right)^2 x, & m - \text{nieparzyste} \end{cases}$$

Politechnika Wrocławska

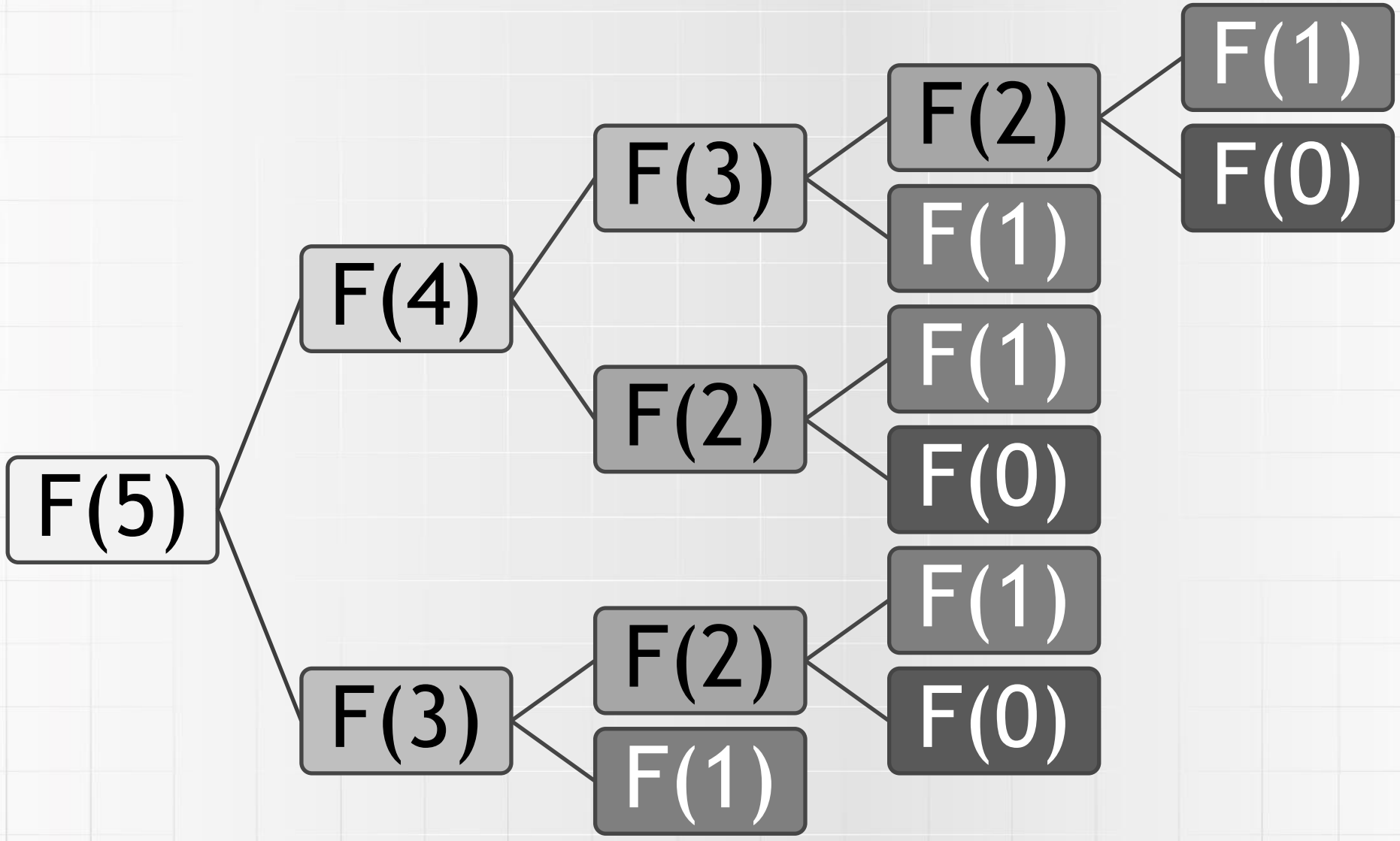
Ciąg Fibonacciego

$$F(n) = \begin{cases} 0 & \text{dla } n = 0, \\ 1 & \text{dla } n = 1, \\ F(n-1) + F(n-2) & \text{dla } n > 1. \end{cases}$$

Politechnika Wrocławska

Ciąg Fibonacciego

Wywołania rekurencyjne



Politechnika Wrocławska

Podsumowanie

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

- Funkcje rekurencyjne
  - Co to jest rekurencja?
  - Warunek stopu
  - Przykłady algorytmów rekurencyjnych