

Podstawy programowania W11OPA-SI0072G

Wstęp do programowania W11FTE-SI0141WL

Wstęp do programowania W11IKW-SI0080WL

rok akademicki 2024/25
semestr letni

Wykład 6

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 221

Plan prezentacji (1)

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

Plan prezentacji (2)

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

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

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
```

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 X
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         #wyświetl wynik
17         print(number)
18
19 main()
20
```

Importowanie modułów

```
01_import_random_randint.py X
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         #wyświetl wynik
17         print(number)
18
19 main()
20
```


Wybrane funkcje z modułu `random`

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

Importowanie modułów

```
01_import_random_randint.py X
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()
20
```

Wybrane funkcje z modułu random

```
01_import_random_randint.py X 02_randint.py X
1  # Program symuluje serię rzutów monetą.
2  # Program wykorzystuje funkcję randint()
3  # z modułu random.
4
5  import random #import modułu
6
7  HEAD = 1      #stała oznaczająca awers (orzeł)
8  TAIL = 2      #stała oznaczająca rewers (reszka)
9  TOSSES = 10   #stała określająca liczbę rzutów
10
11 def main():
12     # w każdym rzucie
13     for toss in range(TOSSES):
14         # wylosuj liczbę (1 lub 2)
15         # 1 oznacza awers (orzeł)
16         if random.randint(HEAD, TAIL) == HEAD:
17             print('orzeł')
18         # w przeciwnym wypadku to rewers (reszka)
19         else:
20             print('reszka')
21
22 main()
```

Wybrane funkcje z modułu `random`

Inne funkcje dostępne w module `random`:

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

Wybrane funkcje z modułu random

```
01_import_random_randint.py X 02_randint.py X 03_random.py X
1  # Program pokazuje różne funkcje z modułu random.
2
3  import random
4
5  def main():
6      #randrange - zwraca losową liczbę z listy
7      print(random.randrange(6))          #losowa wartość od 0 do 5
8      print(random.randrange(7,10))       #losowa wartość od 7 do 9
9      print(random.randrange(0,101,10))   #losowa wartość z listy od 0 co 10 do 100
10     #random - zwraca losową liczbę z przedziału [0.0, 1.0)
11     print(random.random())
12     #uniform - zwraca losową liczbę z przedziału [a, b]
13     print(random.uniform(3.0,5.0))      #losowa wartość z przedziału [3, 5]
14
15 main()
16
```

Wybrane funkcje z modułu `random`

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

Wybrane funkcje z modułu random

```
01_import_random_randint.py X 02_randint.py X 03_random.py X 04_random.py X
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()
15
```

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 X
1  # Program pokazuje działanie funkcji sqrt()
2  # z modułu math.
3
4  #importowanie modułu math
5  import math
6
7  def main():
8      #wczytanie liczby rzeczywistej
9      number = float(input('Podaj Liczbę: '))
10     #obliczenie pierwiastka
11     #funkcja sqrt() z biblioteki math
12     square_root = math.sqrt(number)
13     #wyświetlenie wyników
14     print('Pierwiastek kwadratowy liczby',
15           number, 'wynosi', square_root)
16
17     main()
```

Wybrane funkcje z modułu math

```
05_import_math.py X 06_math_pi.py X
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)
13
```

Wybrane funkcje z modułu `math`

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

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

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

Tworzenie własnych modułów

```
circle.py X rectangle.py X 07_geometry.py X
1  # Moduł circle.py
2  # Zawiera definicje funkcji służących do obliczania:
3  # pola i obwodu koła oraz do wczytywania promienia.
4
5  # import modułu math - wykorzystywana wartość math.pi
6  import math
7
8  # Funkcja oblicza pole koła.
9  # I: promień
10 # P: oblicza pole z zależności: pole = pi*r^2
11 # O: obliczona wartość pola
12 def area(radius):
13     return math.pi * radius ** 2
14
15 # Funkcja oblicza obwód koła.
16 # I: promień
17 # P: oblicza obwód z zależności: obwód = 2*pi*r
18 # O: obliczona wartość obwodu
19 def circumference(radius):
20     return 2 * math.pi * radius
21
22 # Funkcja pobiera promień koła od użytkownika.
23 # I: brak
24 # P: wczytanie liczby rzeczywistej podanej przez użytkownika
25 # O: promień
26 def get_radius():
27     return float(input('Podaj promień: '))
28
```

Tworzenie własnych modułów

```
circle.py X rectangle.py X 07_geometry.py X
1  # Moduł rectangle.py
2  # Zawiera definicje funkcji służących do obliczania:
3  # pola i obwodu prostokąta oraz
4  # do wczytywania jego wymiarów.
5
6  # Funkcja oblicza pole prostokąta.
7  # I: długość i szerkość
8  # P: oblicza pole z zależności: pole = długość*szerkość
9  # O: obliczona wartość pola
10 def area(length, width):
11     return length * width
12
13 # Funkcja oblicza obwód prostokąta.
14 # I: długość i szerkość
15 # P: oblicza obwód z zależności: obwód = 2*(długość+szerkość)
16 # O: obliczona wartość obwodu
17 def perimeter(length, width):
18     return 2 * (length + width)
19
20 # Funkcja pobiera wymiary prostokąta od użytkownika.
21 # I: brak
22 # P: wczytanie liczby rzeczywistej podanej przez użytkownika
23 # O: (długość, szerkość)
24 def get_dimensions():
25     length = float(input('Podaj długość prostokąta: '))
26     width = float(input('Podaj szerokość prostokąta: '))
27     return (length, width)
28
```

Tworzenie własnych modułów

```
circle.py X rectangle.py X 07_geometry.py X
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
```


Tworzenie własnych modułów

```
circle.py X rectangle.py X 07_geometry.py X
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
```

Tworzenie własnych modułów

```
circle.py X rectangle.py X 07_geometry.py X
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:
39             radius = circle.get_radius()
40             print('Pole koła wynosi', circle.area(radius))
41         elif choice == CIRCUMFERENCE_CHOICE:
42             radius = circle.get_radius()
43             print('Obwód koła wynosi', circle.circumference(radius))
44         elif choice == AREA_RECTANGLE_CHOICE:
45             (length, width) = rectangle.get_dimensions()
46             print('Pole prostokąta wynosi', rectangle.area(length, width))
47         elif choice == PERIMETER_RECTANGLE_CHOICE:
48             (length, width) = rectangle.get_dimensions()
49             print('Obwód prostokąta wynosi', rectangle.perimeter(length, width))
50         elif choice == QUIT_CHOICE:
51             print('Zakończenie działania programu.')
52         else:
53             print('Błąd: nieprawidłowa opcja.')
```

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

#pole koła
#obwód koła
#pole prostokąta
#obwód prostokąta
#koniec programu
#wartość spoza listy

Tworzenie własnych modułów

```
circle.py X rectangle.py X 07_geometry.py X
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
```

Tworzenie własnych modułów

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

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

Funkcje rekurencyjne

```
recursion_endless.py X recursion_01.py X
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

Funkcje rekurencyjne

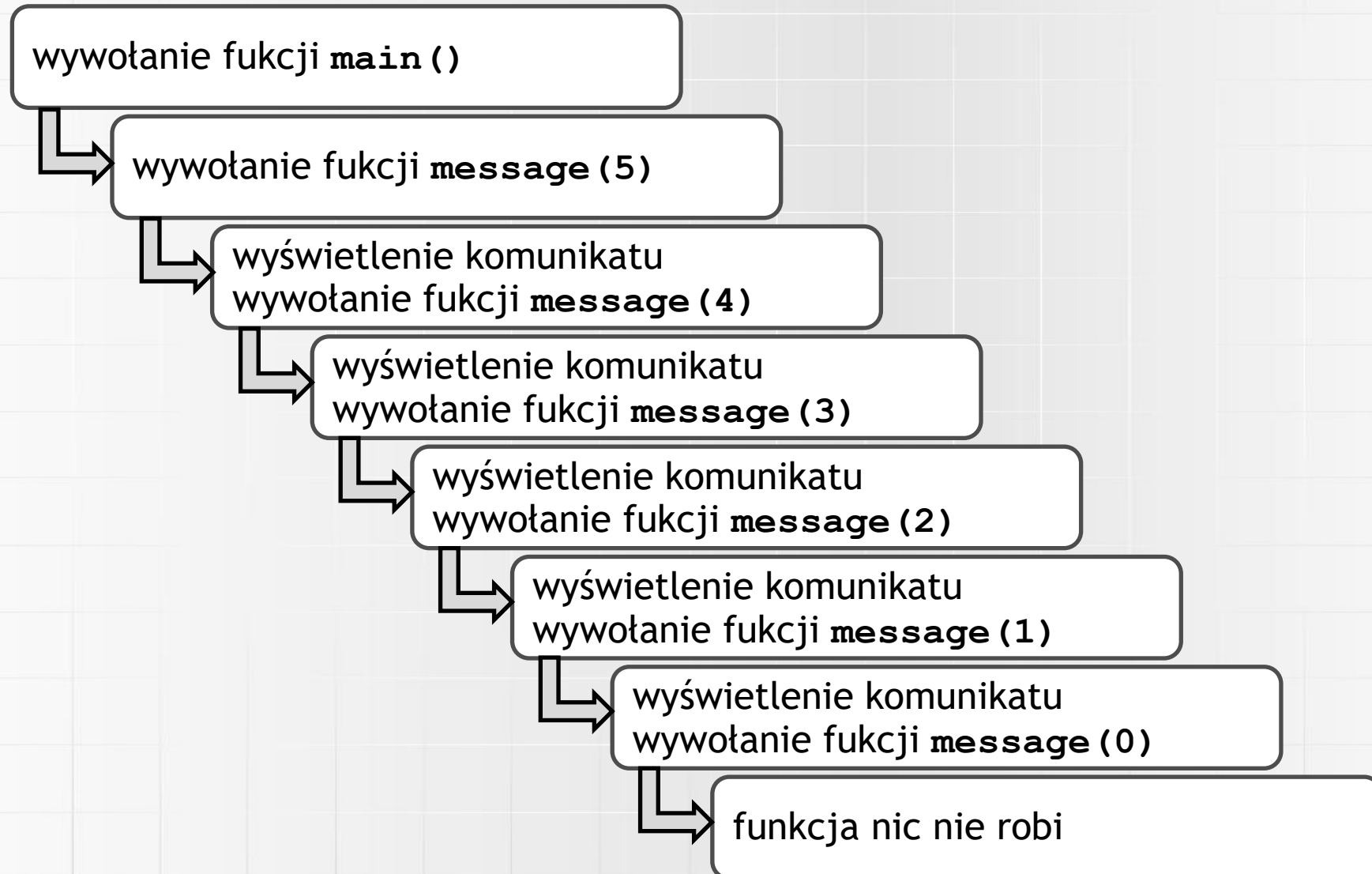
```
recursion_endless.py X recursion_01.py X
1  #Program pokazuje przykładową funkcję rekurencyjną.
2
3  def main():
4      message(5)
5
6  #I :liczba całkowita określająca liczbę
7  #   wywołań rekurencyjnych
8  #P :jeśli liczba wywołań jest dodatnia
9  #   wyświetl informację
10 #   wywołaj rekurencyjnie funkcję message()
11 #   z liczbą wywołań mniejszą o jeden
12 #0 :brak
13 def message(times):
14     if times > 0:
15         print('To jest funkcja rekurencyjna')
16         message(times - 1)
17
18 main()
19
```


Funkcje rekurencyjne

```
recursion_endless.py X recursion_01.py X
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 #0 :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

Funkcje rekurencyjne



Funkcje rekurencyjne

```
recursion_endless.py X recursion_01.py X recursion_02.py X
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 jest dodatnia
9  #   wyświetl informację
10 #   wywołaj rekurencyjnie funkcję message()
11 #   z liczbą wywołań mniejszą o jeden
12 #0 :brak
13 def message(times):
14     if times > 0:
15         print('Wartość parametru times', times)
16         message(times - 1)
17
18 main()
```

Funkcje rekurencyjne

```
recursion_endless.py X recursion_01.py X recursion_02.py X
1  #Program pokazuje przykładową funkcję rekurencyjną.
2
3  def main():
4      message(5)
5
6  #I :liczba całkowita określająca
7  #   wywołań rekurencyjnych
8  #P :jeśli liczba wywołań jest
9  #   wyświetl informację
10 #   wywołaj rekurencyjnie
11 #   z liczbę wywołań m
12 #0 :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
```

Funkcje rekurencyjne

```
recursion_endless.py X recursion_01.py X recursion_02.py X recursion_03.py X
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 liczbą 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
```

Funkcje rekurencyjne

```
recursion_endless.py X recursion_01.py X recursion_02.py X recursion_03.py X
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 do
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

Funkcje rekurencyjne

Silnia

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

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

Funkcje rekurencyjne

Silnia

```
recursion_endless.py X recursion_01.py X recursion_02.py X recursion_03.py X factorial.py X
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 #0 :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()
```


Funkcje rekurencyjne

Silnia

```
recursion_endless.py X recursion_01.py X recursion_02.py X recursion_03.py X factorial.py X factorial_02.py X
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()
```

Funkcje rekurencyjne

Silnia

```
recursion_endless.py X recursion_01.py X recursion_02.py X recursion_03.py X factorial.py X factorial_02.py X
1 #Program demonstruje działanie funkcji faktorial
2 #obliczającej wartość silni
3
4 def main():
5     print('5! =', factorial(5))
6
7 def factorial(n, level = 0):
8     print(f"level = {level}")
9     if n == 0:
10         print(f"level = {level}")
11         return 1
12     else:
13         x = n*factorial(n-1)
14         print(f"level = {level}")
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
```

Algorytmy rekurencyjne

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

Algorytm Euklidesa

zakładając, że $m \leq n$

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

Obliczanie potęgi

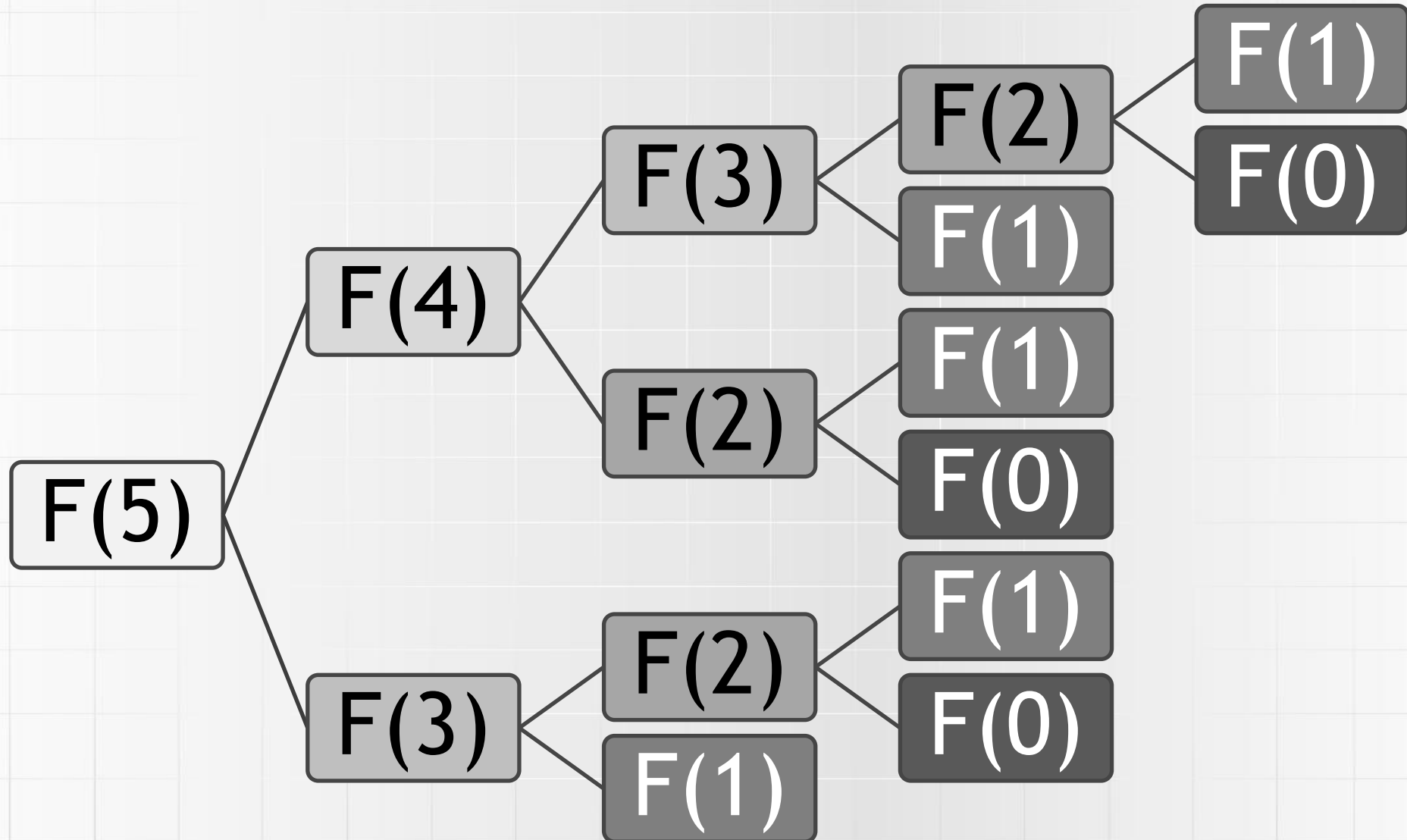
$$x^m = \begin{cases} x, & \text{dla } m = 1 \\ \left(x^{m/2}\right)^2, & m - \text{parzyste} \\ \left(x^{(m-1)/2}\right)^2 x, & m - \text{nieparzyste} \end{cases}$$

Ciąg Fibonacciego

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

Ciąg Fibonacciego

Wywołania rekurencyjne



Podsumowanie

- Importowanie modułów biblioteki standardowej
- Znajomość wybranych funkcji z modułów `random` oraz `math`
- Funkcje rekurencyjne
 - Co to jest rekurencja?
 - Warunek stopu
 - Przykłady algorytmów rekurencyjnych