Politechnika Wrocławska

Programowanie Procduralne
Wykład #6(W#13) — Fizyka

HR EXCELLENCE IN RESEARCH

Janusz Andrzejewski

Wstęp

- „Dziwne liczby”
- Moduł math
  - „Dziwne liczby”
  - Liczby zmiennoprzecinkowe
  - Liczby całkowite
- Funkcje wyższego rzędu
  - `filter()`
  - `map()`
    - `reduce()` – moduł `functools`
    - `filterfalse()` – moduł `itertools`

Plan czyli czego się będziemy uczyli?

- W#1 – Podsumowanie
- – Typy mutowalne oraz niemutowalne a przekazywanie argumentów do funkcji.
- – Zasięg zmiennych — zmienne lokalne, nielokalne oraz globalne.
- – Argumenty funkcji: pozycyjne, nazwane, domyślne, rozpakowywanie, słownikowe
- – Funkcje: „first citizen object”.
- – Funkcje: funkcje wyższego rzędu.
- – Funkcje anonimowe.
- – Dopełnienia funkcji.
- – Zastosowania funkcji.
- Laboratoria: od teorii do praktyki czyli rozwiązywanie problemów związanych z omówionymi zagadnieniami na wykładzie.

„Dziwne liczby”

- Są to liczby typu **`float`**
- Nieskończoność – **`float('inf')`** – „liczba” zawsze większa od dowolnej liczby zmiennoprzecinkowej
- Ujemna nieskończoność – **`float('-inf')`**– „liczba” zawsze mniejsza od dowolnej liczby zmiennoprzecinkowej
- Nie liczba (not a number) - **`float('nan')`** – stosowana do oznaczenia że niektóre operacje arytmetyczne są niedozwolone

```
1 x=float('inf')
2 print("x= ",x)
3 print("Typ: ",type(x))
```

```
x=  inf
Typ:  <class 'float'>
```

```
1 y=float("nan")
2 print("y= ",y)
3 print("Typ: ",type(y))
```

```
y=  nan
Typ:  <class 'float'>
```

„Dziwne liczby” — własności

```
1 a=float("inf")
2 b=float("inf")
3 x=float("nan")
4 y=float("nan")
```

```
1 print("inf+inf:", a+b)
```

```
inf+inf: inf
```

```
1 print("inf-inf:", a-b)
```

```
inf-inf: nan
```

```
1 print("2*inf:", 2*a)
```

```
2*inf: inf
```

```
1 print("-2*inf:", -2*a)
```

```
-2*inf: -inf
```

```
1 print("inf+3:", a+3)
```

```
inf+3: inf
```

```
1 print("1/inf:", 1/a)
```

```
1/inf: 0.0
```

```
1 print("-1/inf:", -1/a)
```

```
-1/inf: -0.0
```

```
1 print("inf/inf:", b/a)
```

```
inf/inf: nan
```

```
1 print("inf == inf:", a==b)
```

```
inf == inf: True
```

„Dziwne liczby” — własności

```
1 a=float("inf")
2 b=float("inf")
3 x=float("nan")
4 y=float("nan")
```

```
1 print("nan+nan:", x+y)
```

```
nan+nan: nan
```

```
1 print("3*nan :",3*x)
```

```
3*nan : nan
```

```
1 print("3/nan :",3/x)
```

```
3/nan : nan
```

```
1 print("nan/nan :",y/x)
```

```
nan/nan : nan
```

```
1 print("nan==nan :",y==x)
```

```
nan==nan : False
```

```
1 print("1/-0.0 :", 1/-0.0)
```

```
----------------------------------------
ZeroDivisionError                         T
Cell In[18], line 1
----> 1 print("1/-0.0 :", 1/-0.0)

ZeroDivisionError: float division by zero
```

„Dziwne liczby” — moduł math

```
1 import math
2 a = math.inf
3 x = math.nan
4 print('Czy "liczba" inf skończona ?', math.isfinite(a))
5 print('Czy "liczba" nan skończona ?', math.isfinite(x))
6 print('Czy "liczba" inf nieskonczona ?',math.isinf(a))
7 print('Czy "liczba" nan nieskonczona ?',math.isinf(x))
8 print('Czy "liczba" inf nienormalna ?',math.isnan(a))
9 print('Czy "liczba" nan nienormalna ?',math.isnan(x))
```

```
Czy "liczba" inf skończona ? False
Czy "liczba" nan skończona ? False
Czy "liczba" inf nieskonczona ? True
Czy "liczba" nan nieskonczona ? False
Czy "liczba" inf nienormalna ? False
Czy "liczba" nan nienormalna ? True
```

Liczby rzeczywiste (1/2):

```
1 import sys
2 print(sys.float_info)
```

```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308,
min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant
_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

Liczba epsilon – najmniejsza liczba zmiennoprzecinkowa dla której prawdziwa jest relacja:

1.0+epsilon > 1.0

```
1 print(sys.float_info.epsilon)
```

```
2.220446049250313e-16
```

Liczby rzeczywiste (2/2):

```
1 print("Max float > inf?:", sys.float_info.max > float("inf"))
2 print("-Max float < -inf?:", -sys.float_info.max < float("-inf"))
```

```
Max float > inf?: False
-Max float < -inf?: False
```

```
1 print("Max float > nan?:", sys.float_info.max > float("nan"))
2 print("Max float < nan?:", sys.float_info.max < float("nan"))
```

```
Max float > nan?: False
Max float < nan?: False
```

```
1 x = float(1.0e308)
2 a = float(1.0e309)
3 print("x=",x)
4 print("a=",a)
```

```
x= 1e+308
a= inf
```

```
1 x = sys.float_info.max
2 x2=x+x*1.0e-10
3 x3=x+1.0e100
4 print("x2=", x2)
5 print("x3=", x3)
```

```
x2= inf
x3= 1.7976931348623157e+308
```

Liczby zmiennoprzecinkowe — moduł math

Następnik liczby x: najmniejsza zmiennoprzecinkowa liczba taka dla której pomiędzy liczbą x a jej następnikiem nie ma innej binarnej reprezentacji zmiennoprzecinkowej liczby x

`math.nextafter(x, y)` - następnik liczby x w kierunku liczby y

`math.copysign(x, y)` – zwraca moduł liczby x ze znakiem liczby y

```
1 math.nextafter(1, 2)
```

```
1.0000000000000002
```

```
1 math.nextafter(2023, 2), math.nextafter(2023, 2024)
```

```
(2022.9999999999998, 2023.0000000000002)
```

```
1 math.copysign(1.23, 1), math.copysign(-1.23, 1)
```

```
(1.23, 1.23)
```

```
1 math.copysign(1.23, -1), math.copysign(-1.23, -1)
```

```
(-1.23, -1.23)
```

```
1 math.copysign(1.0, -0.0), math.copysign(1, 0.0)
```

```
(-1.0, 1.0)
```

math — liczby zmiennoprzecinkowe

math.ulp(x) - dokładność liczby x, tzn. najmniejsza liczba zmiennoprzecinkowa dla której zachodzi relacja:
abs(x)+math.ulp(x) > abs(x)

```
1 math.ulp(1.0) #epsilon
```

```
2.220446049250313e-16
```

```
1 math.ulp(10.0)
```

```
1.7763568394002505e-15
```

```
1 math.ulp(-100.0), math.ulp(100.0)
```

```
(1.4210854715202004e-14, 1.4210854715202004e-14)
```

math — liczby zmiennoprzecinkowe

math.floor(x) - największa liczba całkowita mniejsza lub równa x („przybliżenie do liczby całkowitej w kierunku -inf”)
math.ceil(x) - najmniejsza liczba całkowita większa lub równa **x** („przybliżenie do liczby całkowitej w kierunku +inf”)
math.trunc(x) - obcina część ułamkową zwracając liczbę całkowitą („przybliżenie do liczby całkowitej w kierunku zero”)

```
1 math.floor(-2.34), math.floor(2.34)
```

```
(-3, 2)
```

```
1 math.ceil(-2.34), math.ceil(2.34)
```

```
(-2, 3)
```

```
1 math.trunc(-2.34), math.trunc(2.34)
```

```
(-2, 2)
```

math — liczby zmiennoprzecinkowe

`math.isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)` – zwraca True jeśli dwie liczby są równe wg wyrażenia:

abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)

```
1 a=2.34
2 b=math.nextafter(a, math.inf)
3 print("a, b", a, b)
4 print("Czy równe?", math.isclose(a, b))
```

```
a, b 2.34 2.3400000000000003
Czy równe? True
```

```
1 x=a+math.ulp(a)*1.0e7
2 print("a, x", a, x)
3 print("Czy równe?", math.isclose(a, x))
```

```
a, x 2.34 2.340000004440892
Czy równe? False
```

`math` — liczby całkowite

`math.comp(n, k)` — oblicza symbol Newtona n po k, czyli n!/(k!*(n-k)!), oznacza także na ile sposobów można wybrać k elementów ze zbioru n-elementowego bez uwzględnienia kolejności

math.perm(n, k=None) — oblicza n!/(n-k)! oraz gdy k=None oblicza n!, oznacza także na ile sposobów można wybrać k lementów ze zbioru n-elementowego bez powtórzeń uwzględniając kolejność elementów. Zalecany sposób obliczania silni liczby n (funkcja **math.factorial(n)** — jest przestarzała)

math.gcd(*liczby_całkowite) — oblicza największy wspólny dzielnik z liczb całkowitych przekazanych jako argumenty do funkcji

math.lcm(*liczby_całkowite) — oblicza najmniejszą wspólną wielkorotność liczb całkowitych przekazanych jako argumenty do funkcji

Funkcja: filter (1/2):

Nagłówek :

```
filter(func, obiekt_iterowalny)
```

func - funkcja jednej zmiennej której wartością jest `True` lub `False`.
Wartością funkcji `filter` jest obiekt generatora składający się tylko z tych elementów `obiekt_iterowalny` dla których funkcja `func` ma wartość `True`

```
1 import math
2 poleKola = (math.pi*x**2 for x in range(4))
3 maleKola = filter(lambda x: x < 5, poleKola)
4 print("Małe koła - generator:",maleKola)
5 print("Małe koła - lista:",list(maleKola))
```

```
Małe koła - generator: <filter object at 0x7f764c32e980>
Małe koła - lista: [0.0, 3.141592653589793]
```

Funkcja: filter (2/2):

```
1 import math
2 poleKola = (math.pi*x**2 for x in range(4))
3 maleKola = filter(lambda x: x < 5, poleKola)
4 dużeKola = filter(lambda x: x >= 5, poleKola)
5 print("Małe koła:",list(maleKola))
6 print("Duże kola:", list(dużeKola))
```

```
Małe koła: [0.0, 3.141592653589793]
Duże kola: []
```

```
1 import math
2 poleKola = list(math.pi*x**2 for x in range(4))
3 maleKola = filter(lambda x: x < 5, poleKola)
4 dużeKola = filter(lambda x: x >= 5, poleKola)
5 print("Małe koła:",list(maleKola))
6 print("Duże kola:", list(dużeKola))
```

```
Małe koła: [0.0, 3.141592653589793]
Duże kola: [12.566370614359172, 28.274333882308138]
```

Funkcja: map (1/2):

Nagłówek :

`map(func, *obiekty_iterowalne)`

`func` – funkcja tylu zmiennych ile jest przekazanych obiektów iterowalnych do funkcji `map`

`*obiekty_iterowalne` – dowolna ilość obiektów iterowalnych. Wartością funkcji `map` jest generator składający się z wartości funkcji `func` zastosowanych „odpowiednio” do `*obiekty_iterowalne`

```
1 def suma(x, y):
2     return x+y
3
4 x = [1, 2, 3]
5 y = [-1, 2.3, -3.33]
6 wyn=map(suma, x, y)
7 print(list(wyn))
8
```

```
[0, 4.3, -0.33000000000000007]
```

Funkcja: map (2/2):

```
1 poleKola = list(math.pi*x**2 for x in range(1, 4))
2 wyn = list(map(round, poleKola, range(1, 7)))
3 print(wyn)
```

```
[3.1, 12.57, 28.274]
```

```
1 poleKola = list(math.pi*x**2 for x in range(1, 7))
2 wyn = list(map(round, poleKola, range(1, 7)))
3 print(wyn)
```

```
[3.1, 12.57, 28.274, 50.2655, 78.53982, 113.097336]
```

Funkcja: reduce (1/3):

- Funkcja dostępna w module `functools`
- **`reduce(func, obiekt_iterowalny, [, initial])`**
  - `func` - funkcja dwu zmiennych
  - `reduce` stosuje funkcje dwu zmiennych dla każdego elementu obiekt_iterowalny,

```
1 from functools import reduce
2 liczby=[1, 2, 3, 4]
3 def suma(x, y):
4     wyn = x + y
5     print(f"{x} + {y} = {wyn}")
6     return wyn
7
8 S=reduce(suma, liczby)
9 print("Suma:",S)
```

```
1 + 2 = 3
3 + 3 = 6
6 + 4 = 10
Suma: 10
```

Funkcja: reduce (2/3):

```
1  from functools import reduce
2  liczby=[1, 2, 3]
3  def suma(x, y):
4      return x+y
5
6  def iloczyn(x, y):
7      return x*y
8
9  S=reduce(suma, liczby)
10 I=reduce(iloczyn, liczby)
11 print("Suma:", S)
12 print("Iloczyn:", I)
```

```
Suma: 6
Iloczyn: 6
```

```
1  S=reduce(suma, liczby, 1)
2  I=reduce(iloczyn, liczby, 2)
3  print("Suma:", S)
4  print("Iloczyn:", I)
```

```
Suma: 7
Iloczyn: 12
```

Funkcja: reduce (3/3):

```
1 from functools import reduce
2
3 def suma(x, y):
4     return x+y
5
6 S = reduce(suma, [])
7 prit("Suma:", 0)
```

```
----------------------------------------------
TypeError                                 Tra
Cell In[71], line 6
      3 def suma(x, y):
      4     return x+y
----> 6 S = reduce(suma, [])
      7 prit("Suma:", 0)

TypeError: reduce() of empty iterable with no initial value
```

```
1 from functools import reduce
2
3 def suma(x, y):
4     return x+y
5
6 S = reduce(suma, [], 0)
7 print("Suma:", 0)
```

```
Suma: 0
```

Funkcja: filterfalse

- Funkcja dostępna w module itertools
- Podobna do standardowej funkcji `filter()` – z tym że zwraca generator dla elementów których funkcja przyjmuje wartość `False`

```
import math
from itertools import filterfalse
def Male(x):
    return x < 5

def Duże(x):
    return not Male(x)

poleKola = list(math.pi*x**2 for x in range(4))
maleKola = filter(Male, poleKola)
dużeKola = filter(Duże, poleKola)
print("Małe koła:",list(maleKola))
print("Duże kola:", list(dużeKola))
dużeKolaInaczej = filterfalse(Male, poleKola)
print("Duże koła inaczej:",list(dużeKolaInaczej))
```

```
Małe koła: [0.0, 3.141592653589793]
Duże kola: [12.566370614359172, 28.274333882308138]
Duże koła inaczej: [12.566370614359172, 28.274333882308138]
```

filter() oraz filterfalse()

Jako funkcji filtrującej w funkcjach **filter()** oraz **filterfalse()** można użyc **None**

```
for i in filter(None, range(4)):
    print(i)
```

```
1
2
3
```

```
from itertools import filterfalse

for i in filterfalse(None, range(4)):
    print(i)
```

```
0
```

filter() oraz filterfalse()

```
1 krotka=(1, 2, 0, "", (), [], {}, "qw")
2 ile=len(tuple((filter(None, krotka))))
3 print(ile)
```

```
3
```

```
1 ile = sum(1 for i in krotka if i)
2 print(ile)
```

```
3
```

```
1 from itertools import filterfalse
2 krotka=(1, 2, 0, "", (), [], {}, "qw")
3 ile=len(tuple((filterfalse(None, krotka))))
4 print(ile)
```

```
5
```

```
1 ile = sum(1 for i in krotka if not i)
2 print(ile)
```

```
5
```

Dziękuję za uwagę