Kolejne ciekawe tematy

- Rekurencja
- Generatory
- Funkcja lambda
- Programowanie obiektowe
- Programowanie współbieżne
- Biblioteki tematyczne
- ...

1

Funkcja rekurencyjna

n! = 1 * 2 * 3 * … * n

$$n! = \begin{cases} 1 & \text{gdy } n<2 \\ (n-1)! * n & \text{gdy } n>=2 \end{cases}$$

```
def silnia(n):
    s = 1
    while n>1:
        s = s*n
        n = n-1
    return s
# end def
```

```
def silnia(n):
  if n<2: return 1
  else:  return silnia(n-1)*n
# end def
```

2

Zamiana na postać dwójkową

```
def bin(n):
    l = []
    while n>0:
        l.append(n%2)
        n = n//2
    # end while
    i = len(l)-1
    while i>=0:
        print(l[i], end="")
        i = i-1
    # end while
# end def
```

```
def bin(n):
    if n>0:
        bin(n//2)
        print(n%2, end="")
    # end if
# end def
```

3

Ciąg Fibonacciego

$$F(n) = \begin{cases} 1 & \text{dla } n < 3 \\ F(n-1)+F(n-2) & \text{dla } n >= 3 \end{cases}$$

1	1	2	3	5	8	13	21				

```
def fib(n):
    a=b=1
    for i in range(1,n):
        c=a+b
        a=b
        b=c
    return a
# end def
```

```
def fib(n):
   if n<3: return 1
   else: return  fib(n-1)+fib(n-2)
# end def
```

4

Waga szalkowa

Problem:

Dany jest zestaw odważników. Jakie ciężary można odważyć z użyciem tych odważników?

Przykład:

```
odw = [1,2,5,10,16,24]

def waga(li,n,p):
    if n==0: return True
    if p==len(li): return False
    return waga(li,n-li[p],p+1) or waga(li,n,p+1)
# end def

for w in range(1,50):
    print(w,waga(odw,w,0))
```

5

Przykład - pary

Problem:

Dana jest tablica/lista z liczbami naturalnymi. Należy policzyć ile jest par elementów o określonym iloczynie.

Przykład:

```
l = [4,1,5,7,9,4,5,9,6,5,3,2,7,6,1,1,7,9,9,1]
```

Są 4 pary o iloczynie 24.

6

Przykład – Licz pary

```
def generuj(n):
    l = []
    for i in range(n): l.append(random.randint(1,9))
    return l
# end def

def licz_pary(l,s):
    licz = 0
    for i in range(len(l)):
        for j in range(i+1,len(l)):
            if l[i]*l[j]==s:
                licz = licz+1
    print("pary",licz)
# end def

t = generuj(20)
licz_pary(t,24)
```

7

Przykład – Licz trójki

```
def licz_trojki(l,s):
    licz = 0
    for i in range(len(l)):
        for j in range(i+1,len(l)):
            for k in range(j+1,len(l)):
                if l[i]*l[j]*l[k]==s:
                    licz = licz+1
    print("trojki",licz)
# end def


t = generuj(20)
licz_trojki(t,24)
```

8

Przykład – Licz n-ki

```
def licz_nki(l,s,n,p):
    global licznik
    if n==1:
        for i in range(p,len(l)):
            if l[i]==s: licznik=licznik+1
    else:
        for i in range(p,len(l)):
            if s%l[i]==0: licz_nki(l,s//l[i],n-1,i+1)
# end def


t = generuj(20)

licznik=0
licz_nki(t,24,4,0)
print("nki",licznik)
```

9

Generatory

```
def podz(n):
  "funkcja zwracająca podzielniki liczby n"
  for p in range(1,n):
    if n%p==0:
      yield p
# end def

for i in podz(120): print(i)
```

10

Generator rekurencyjny

```
def perm(s):
  "Funkcja generuje permutacje liter w napisie s"
  if len(s)==1:
    yield s
  else:
    for p in perm(s[:-1]):
      for i in range(len(s)):
        yield p[:i]+s[-1]+p[i:]
# end def

for a in perm('ABCDE'): print(a)
```

11

Funkcja lambda

lambda arg1, arg2,... argN : expression using arguments

```
def f(x, y):
  return x + y
# end def

>>>f(2,3)
>>>5
```

```
f = lambda x, y: x + y

>>>f(2,3)
>>>5
```

12

Funkcja lambda

```
>>> def make_incrementor (n): return lambda x: x + n

>>> f = make_incrementor(2)
>>> g = make_incrementor(6)

>>> print( f(42), g(42) )
44 48

>>> print( make_incrementor(22)(33) )
55
```

13

Funkcje: filter, map, reduce

```
>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]

>>> print( filter(lambda x: x % 3 == 0, foo) )
[18, 9, 24, 12, 27]

>>> print( map(lambda x: x * 2 + 10, foo) )
[14, 46, 28, 54, 44, 58, 26, 34, 64]

>>> print( reduce(lambda x, y: x + y, foo) )
139
```

14

Biblioteka numpy

```
import numpy as np

A=np.array([[1,2,3,4],[5,6,7,8],[7,8,9,11],[2,3,6,4]])
B=np.array([2,3,5,7])
print(A)
print(B)
x=np.linalg.solve(A,B)
print(x)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 7  8  9 11]
 [ 2  3  6  4]]
[2  3  5  7]
[ 4.625  -9.0   3.125  1.5  ]
```

15

Biblioteka matplotlib

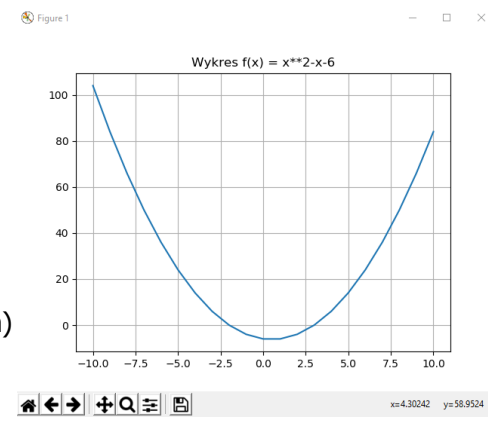
```
import pylab

fun = input("Funkcja: ")

lx = [] # lista argumentów x
ly = [] # lista wartości

for x in range(-10, 11):
    lx.append(x)
    ly.append(eval(fun))

pylab.plot(lx, ly)
pylab.title('Wykres f(x) = '+fun)
pylab.grid(True)
pylab.show()
```



16

Biblioteka turtle

METHOD	PARAMETER	DESCRIPTION
Turtle()	None	It creates and returns a new turtle object
forward()	amount	It moves the turtle forward by the specified amount
backward()	amount	It moves the turtle backward by the specified amount
right()	angle	It turns the turtle clockwise
left()	angle	It turns the turtle counter clockwise
penup()	None	It picks up the turtle's Pen
pendown()	None	Puts down the turtle's Pen
up()	None	Picks up the turtle's Pen
down()	None	Puts down the turtle's Pen
color()	Color name	Changes the color of the turtle's pen
fillcolor()	Color name	Changes the color of the turtle will use to fill a polygon
heading()	None	It returns the current heading
position()	None	It returns the current position
goto()	x, y	It moves the turtle to position x,y
begin_fill()	None	Remember the starting point for a filled polygon
end_fill()	None	It closes the polygon and fills with the current fill color
dot()	None	Leaves the dot at the current position
stamp()	None	Leaves an impression of a turtle shape at the current location
shape()	shapename	Should be 'arrow', 'classic', 'turtle' or 'circle'

17

Programowanie obiektowe

- Obiekt – połączenie danych i operacji na nich wykonywanych, unikatowy egzemplarz danych zdefiniowanych w jego klasie
- Metoda – funkcja określona w definicji klasy
- Klasa – szablon, projekt, prototyp obiektu
- Dziedziczenie – przekazywanie charakterystyki klasy do innych klas

18

Programowanie obiektowe

```
import math

class Circle:                                # class definition statement
    "A 2D circle."                           # documentation string

    def __init__(self, x, y, radius=1): # initialization method
        self.x = x                           # set the attributes
        self.y = y
        self.radius = radius

    def area(self):
        "Return the area of the shape."
        return math.pi * self.radius**2
# end class


i1 = Circle(0, 2)              # '__init__' is called automatically
i2 = Circle(3, 0, 4)

print('i1:', i1.radius, i1.area())
print('i2:', i2.radius, i2.area())

i1: 1 3.14159265359
i2: 4 50.2654824574
```

19

Programowanie współbieżne

```
from threading import Thread
import time,random,sys

class Node(Thread):
  def __init__ (self,n):
    self.n=n
    Thread.__init__(self)

  def run(self):
    for i in range(20):
      time.sleep(random.random())
      print(self.n,end='')
      sys.stdout.flush()
# end class

w1 = Node(0)
w2 = Node(1)
w1.start()
w2.start()
w1.join()
w2.join()
print(”stop”)
```

20

Co dalej?

https://www.nettecode.com/python-czego-zaczac-nauke/

21