Wstęp do Informatyki

Rekurencja

Piotr Pawlik

Problemy z laboratoriów

Zbędne ***range***

Funkcje i ‘kod główny’:

- Zmienne globalne
- main

Zmienne globalne i lokalne

```
x = 5
def fun1() :
    x = 7
    print(x)
```

```
print(x)
>>> 5
fun1()
>>> 7
print(x)
>>> 5
```

Zmienne globalne i lokalne

```
x = 5
def fun1(x) :
    print(x, end=', ')
    x = 7
    print(x)
```

```
print(x)
>>> 5
fun1(x)
>>> 5, 7
print(x)
>>> 5
```

Zmienne globalne i lokalne

```
x = 5
def fun1() :
    global x
    x = 7
    print(x)
```

```
print(x)
>>> 5
fun1()
>>> 7
print(x)
>>> 7
```

Zmienne globalne i lokalne

```
x = 5
def fun1(a) :
    print(a, end=', ')
    a = 7
    print(a)
```

```
print(x)
>>> 5
fun1(x)
>>> 5, 7
print(x)
>>> 5
```

Wywoływanie funkcji

```
x = 5
def fun1(a) :
    print(a, end=', ')
    a = 7
    print(a)
```

```
print(x)
>>> 5
fun1(x)
>>> 5, 7
print(x)
>>> 5
```

Po co są parametry?

```
def pole() :
    return 3.14*r**2
```

```
r=3
print(pole())
r=5
print(pole())
```

```
def używam_globalnej() :
    global r
    r = 77
```

```
używam_globalnej()
.....
print(pole())
```

Po co są parametry?

```
def pole(r) :
    return 3.14*r**2

def obwód(r) :
    return  2*3.14*r
```

```
print(pole(3))

print(pole(5))
```

```
print(pole(7))
print(obwód(7))
```

Main

```
print(pole(7))
print(pole(7))
```

```
r=7
print(pole(r))
print(obwód(r))
```

```
def main():
    r=7
    print(pole(r))
    print(obwód(r))

main()
```

Main

```
print(pole(7))
print(pole(7))
```

```
r=7
print(pole(r))
print(obwód(r))
```

```
def main():
    r=7
    print(pole(r))
    print(obwód(r))

if__name__==’__main__’:
    main()
```

Wzory rekurencyjne

$$n! = n * (n-1)!$$

$$silnia(n) = n * silnia(n-1)$$

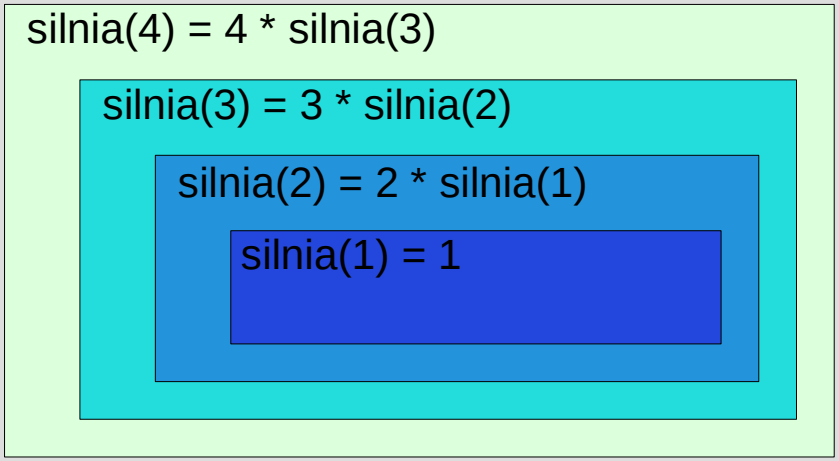
```
def silnia(n):
    return n * silnia(n-1)
```

Wzory rekurencyjne

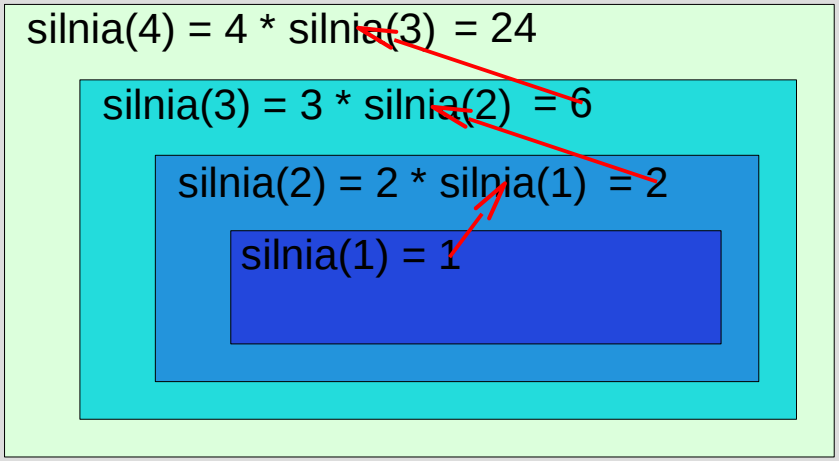
0! = 1
n! = n * (n-1)!

```
def silnia(n):
    if n==0: return 1
    else:    return n * silnia(n-1)
```

Schemat rekurencji



Schemat rekurencji



Rekurencja ogonowa

Rekurencja (prawie) nie wymagająca stosu - może być automatycznie zamieniona na iterację

Warunek: ostatnia wykonywana operacja funkcji musi być albo wywołaniem samej siebie, albo zwróceniem wyniku.

Rekurencja ogonowa

```
def silnia(n):
  if n == 0 :
    return 1
  else :
    return n * silnia(n-1)
```

```
def s(n, acc) :
  if n==0 :
    return acc
  else :
    return s(n-1, n*acc)

def silnia(n):
  return s(n, 1)
```

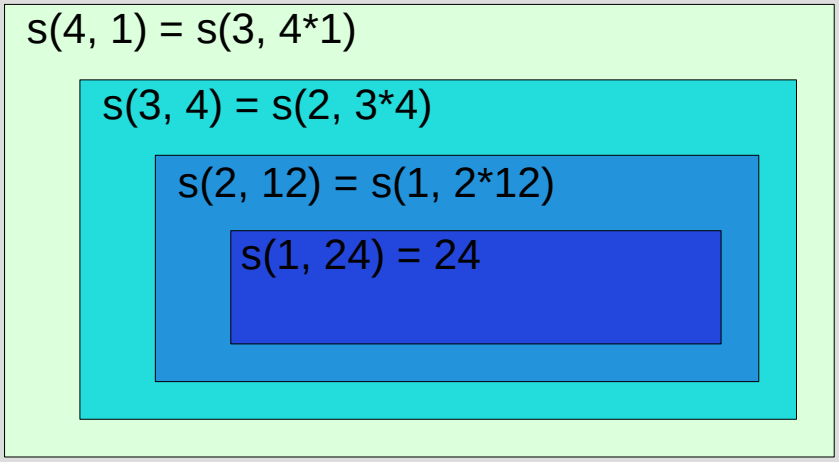
Rekurencja ogonowa

```
def silnia(n) :
  acc=1
  while(n>0) :
    acc=n*acc
    n=n-1
  return acc
```

```
def s(n, acc) :
  if n>0 :
    return s(n-1, n*acc)
  else :
    return acc

def silnia(n):
  return s(n, 1)
```

Schemat rekurencji ogonowej



Procedury

Wprowadzenie standardowych „struktur sterujących”:

- sekwencja
- selekcja (wybór, if-else, switch)
- pętla
- podprogram (jedno wejście, JEDNO wyjście)

Generatory

Drugi, po funkcjach, typ podprogramu w Pythonie. Różnica w tworzeniu polega na użyciu słowa ***yield*** zamiast ***return***. Generator tworzy obiekt iteratorowy – czyli obiekt reprezentujący strumień danych udostępnianych 'na żądanie' (np w pętli for iterator może zastąpić sekwencję)

yield nie kończy podprogramu, lecz go zawiesza. Żądanie kolejnej danej powoduje wznowienie w miejscu ostatniego zawieszenia.

Przykład generatora

```
def square() :
    i=1
    while True :
        yield i**2
        i+=1
```

```
# nieskończona sekwencja
for x in square():
    print(x, end=' ')

# sekwencja 10-ciu kwadratów
it = square()
for i in range(10):
    print( next(it), end=' ')
```

Kolejny przykład generatora

```
def repeater( *arg ) :
    while True :
        for el in arg :
            yield el
```

```
it = repeater('Ala', 'ma', 'kota')
# 5 wywołań:
for i in range(5):
print( next(it), end=' ')
>>> Ala ma kota Ala ma
# Kolejne 5 wywołań:
for i in range(5):
print( next(it), end=' ')
>>> kota Ala ma kota Ala
```

Przekazanie danych do współprogramu

```
def print_name(prefix):
    print("Szukam:", prefix)
    while True:
        name = (yield)
        if prefix in name:
            print(„JEST:",name)
        else:
            print("BRAK")
```

```
corou = print_name("Drogi")
corou.__next__()# to samo
                co # next(corou)
>>> Szukam: Drogi
corou.send("Piotrze")
>>> BRAK
corou.send("Drogi Piotrze")
>>> JEST: Drogi Piotrze
```