RYKA CBT

پدرام شاه صفی
pd.Shahsafi@gmail.com

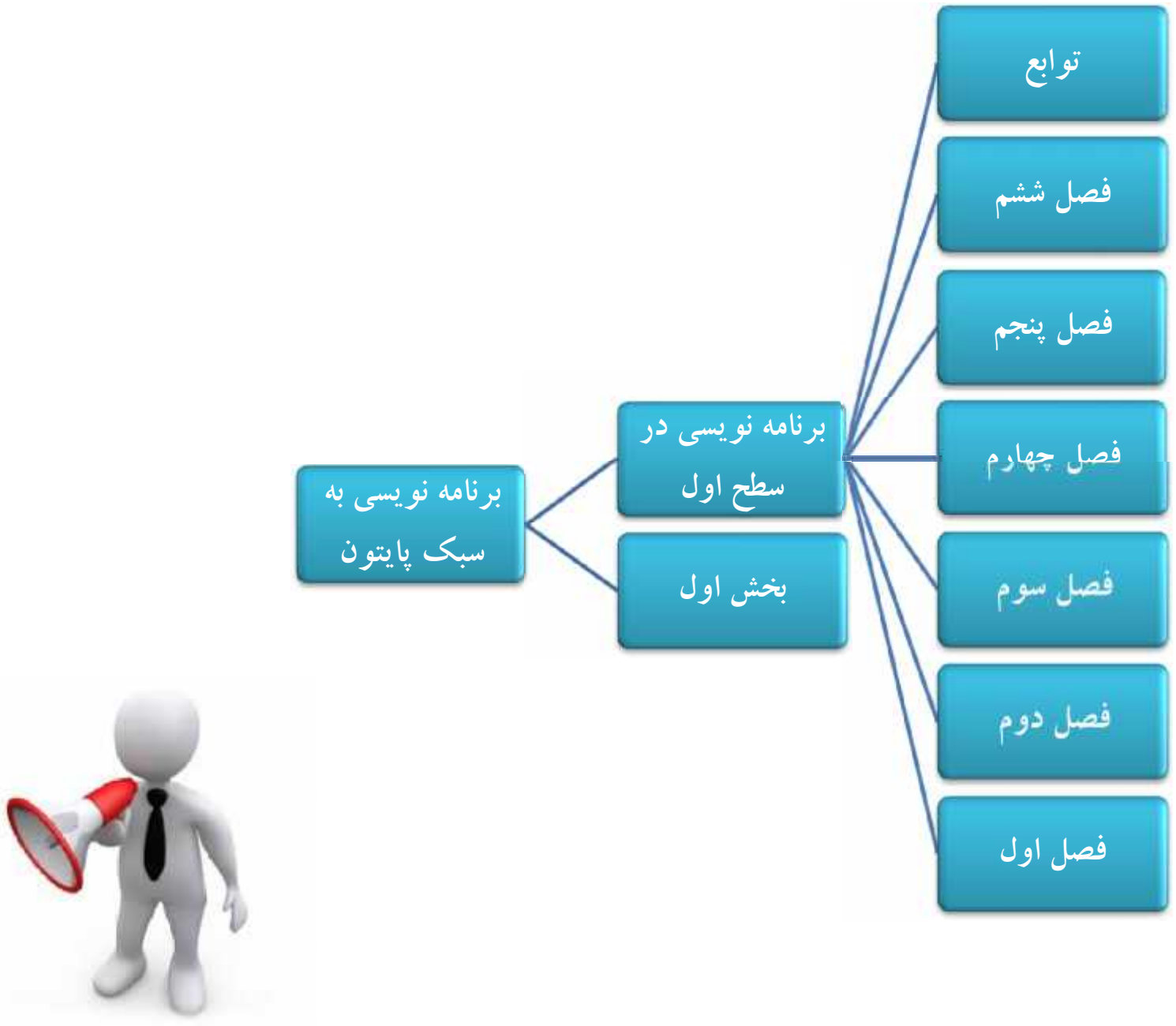
python

به نام پروردگار دانایی

برنامه نویسی به سبک پایتون

پدرام شاه صفی

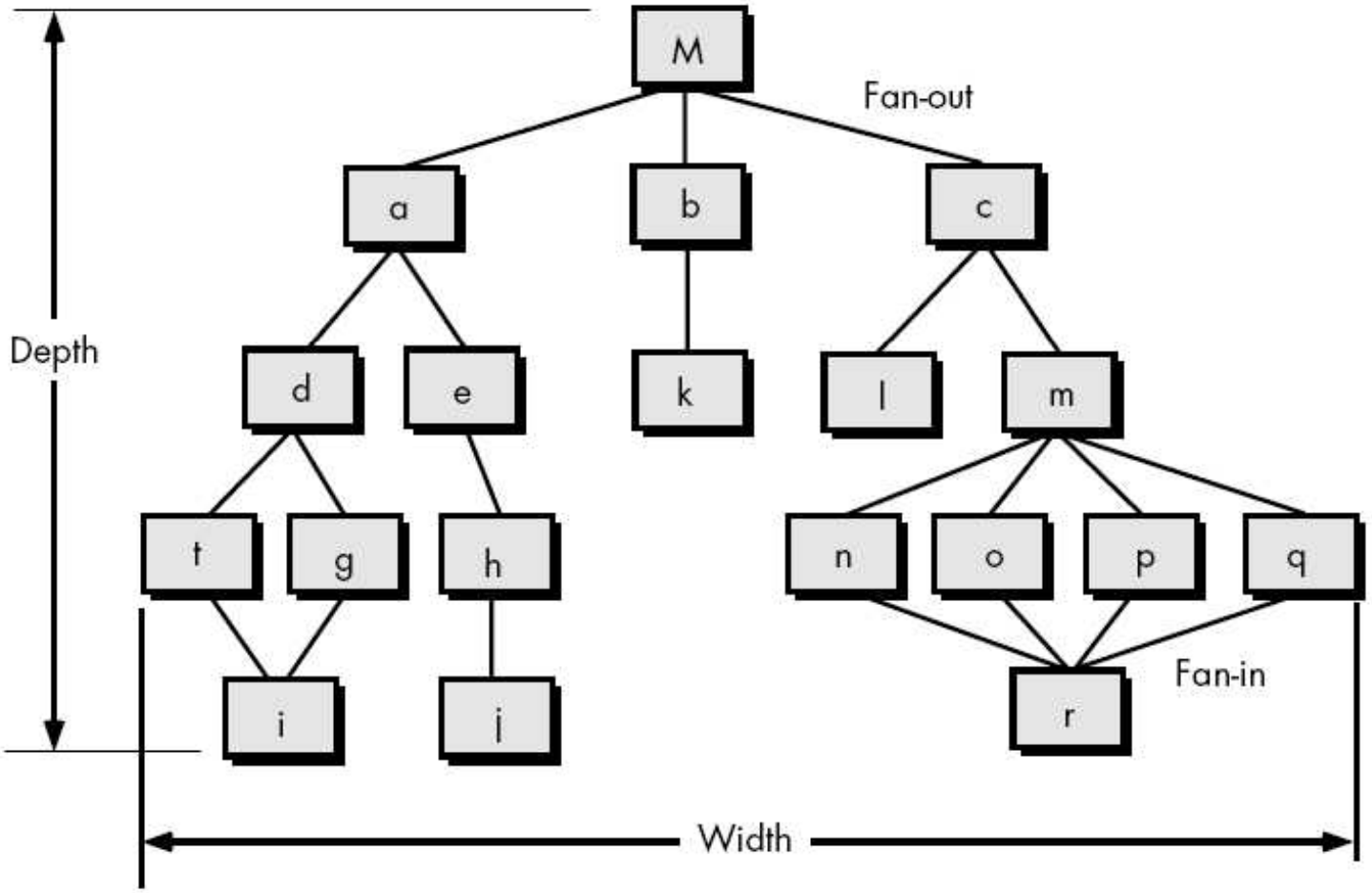
تابستان ۱۳۹۴


برنامه نویسی به سبک پایتون
برنامه نویسی در سطح اول
بخش اول
توابع
فصل ششم
فصل پنجم
فصل چهارم
فصل سوم
فصل دوم
فصل اول

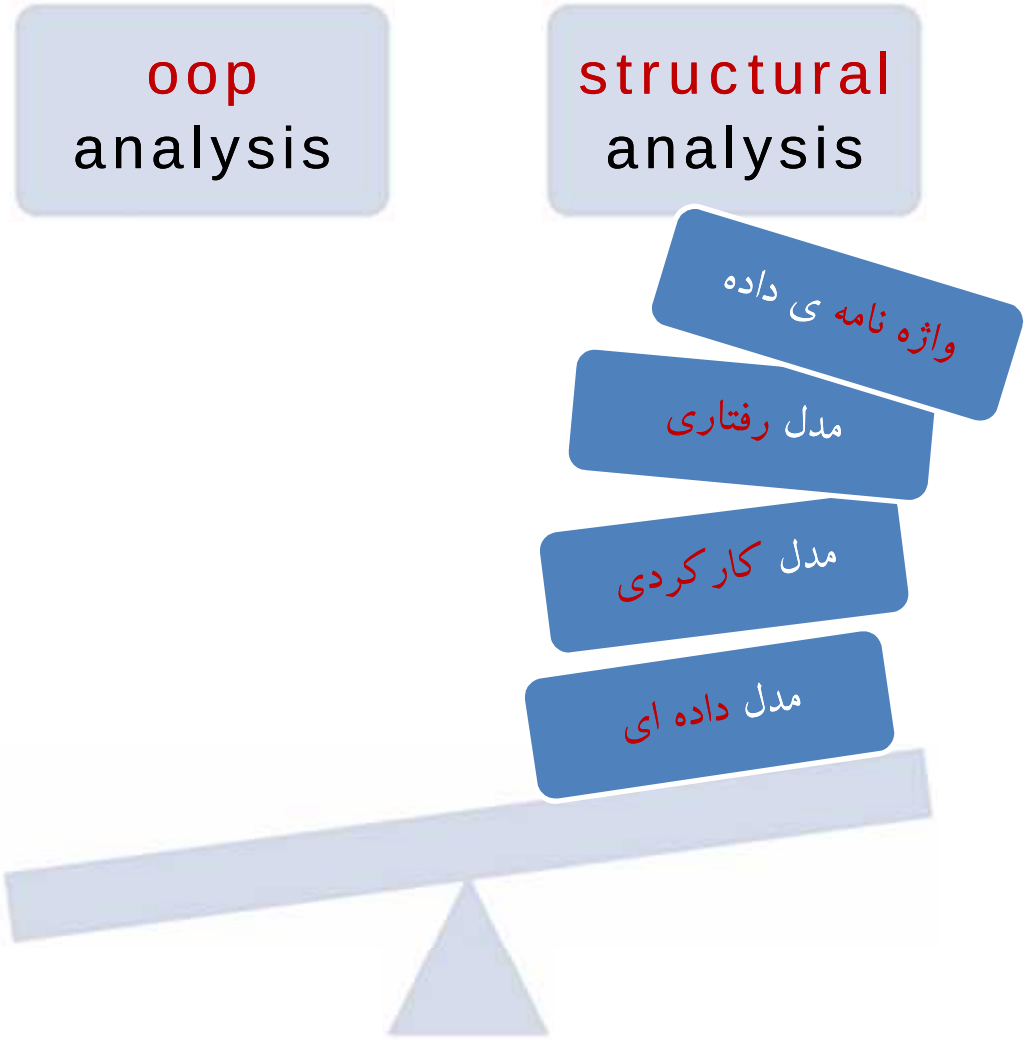
معماری های نرم افزار

Architecture style	***Description***
Client/Server	Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures.
Component-Based Architecture	Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.
Layered Architecture	Partitions the concerns of the application into stacked groups (layers).
Object-Oriented	A design paradigm based on division of responsibilities for an application or system into individual reusable and self-sufficient objects, each containing the data and the behavior relevant to the object.

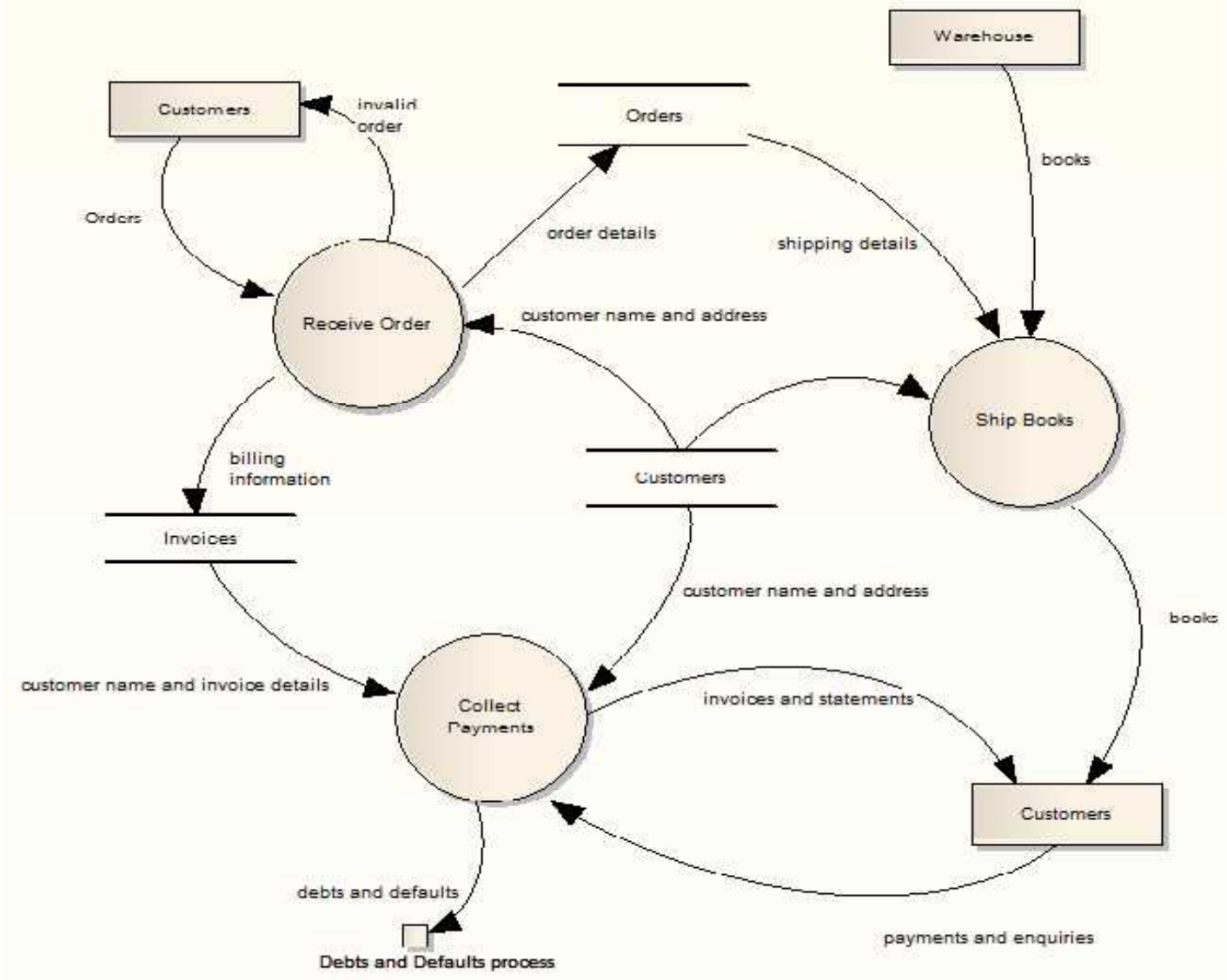
معماری فراخوانی و بازگشت



مدل های تحلیل

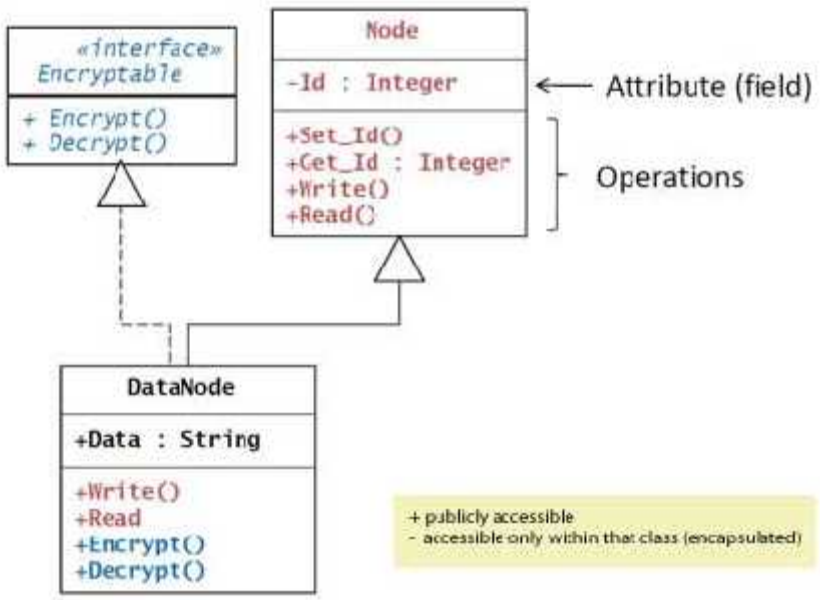


نمودار DFD

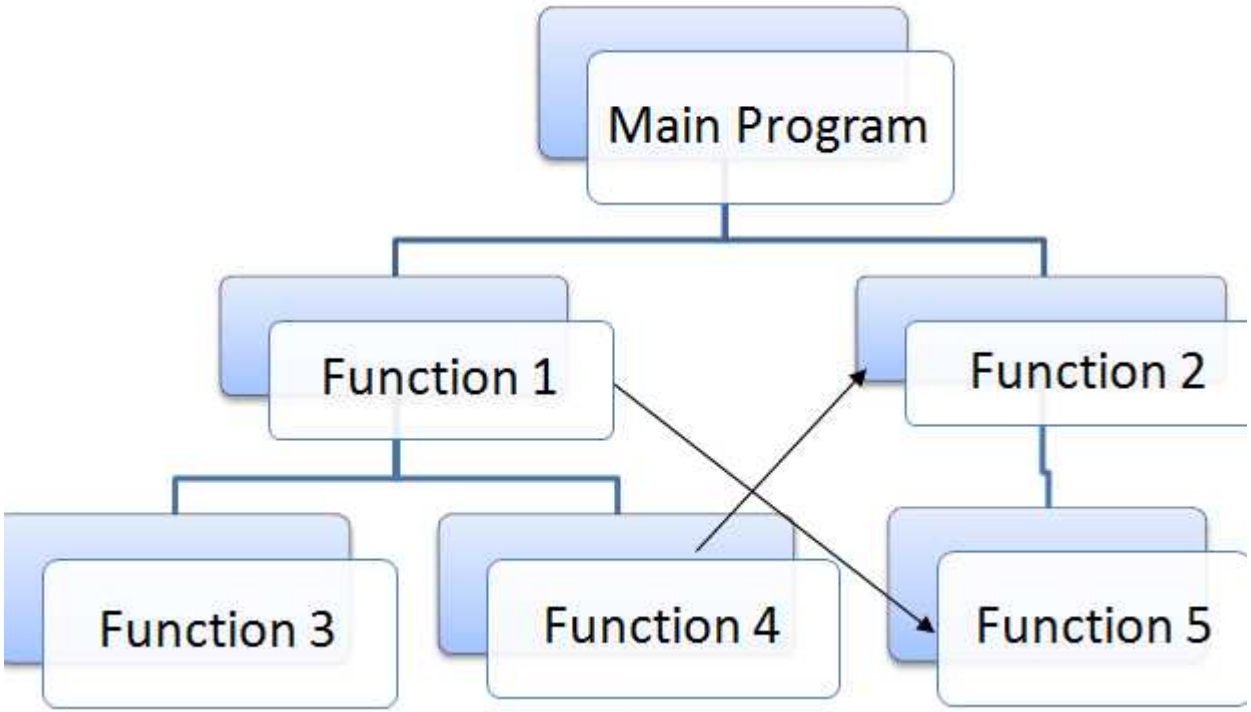

Warehouse
Customers
invalid order
Orders
books
Orders
order details
shipping details
Receive Order
customer name and address
Ship Books
billing information
Customers
Invoices
customer name and address
books
customer name and invoice details
Collect Payments
invoices and statements
Customers
debts and defaults
payments and enquiries
Debts and Defaults process

اتصال مفاهیم

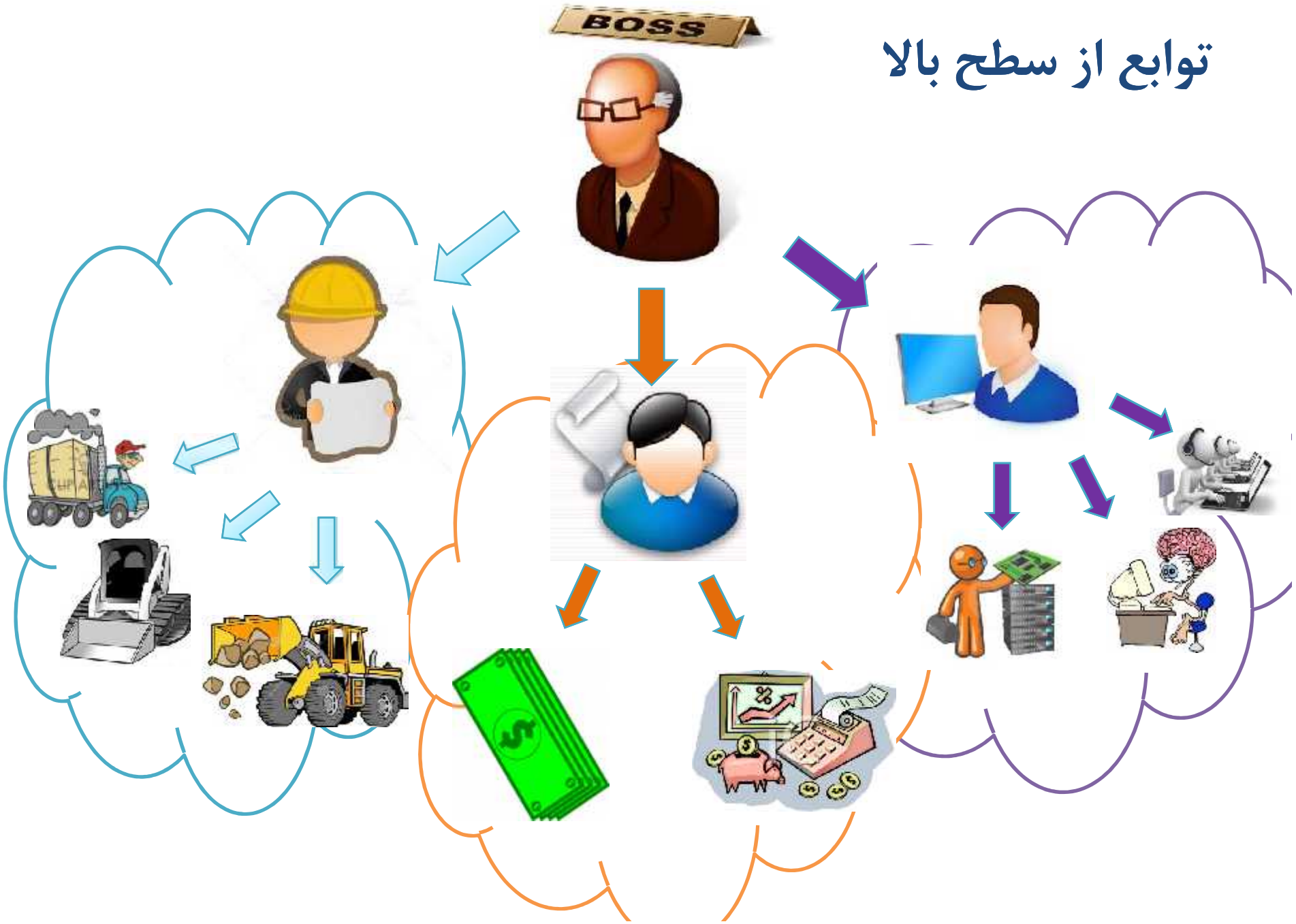
- با StructuralDesign می توان از DFDبه راحتی به معماری Call&Return رسید.



در نهایت


Main Program
Function 1
Function 2
Function 3
Function 4
Function 5

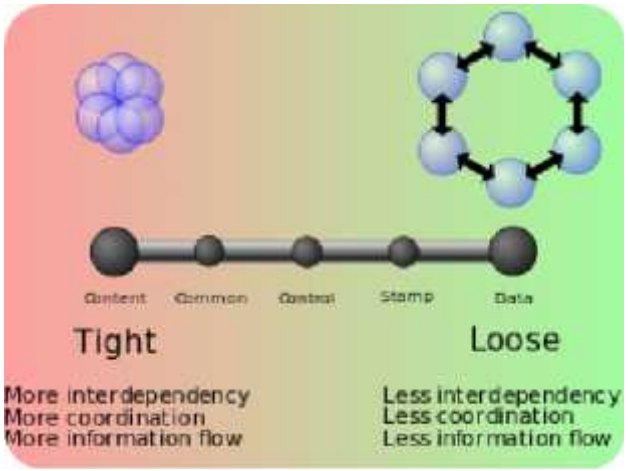
توابع از سطح بالا



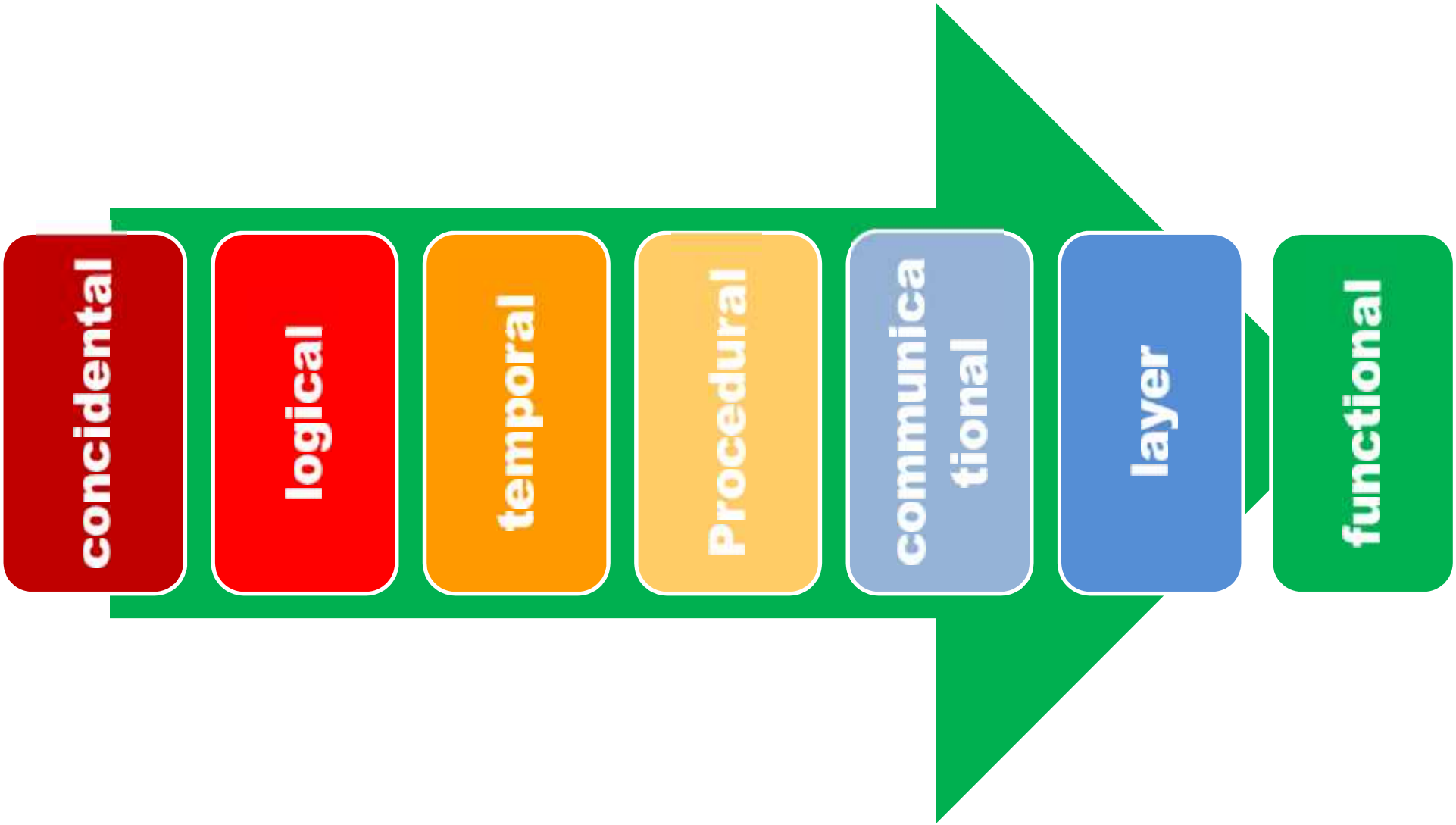
منشاء مشکلات

- **Cohesion**
  - پیوستگی بین ماژول ها را گویند..
  - هر پیمانه یک single task انجام دهد.
  - برای انجام ان کمترین ارتباط را با سایرین داشته باشد.
  - هر چه Cohesion بالاتر باشد بهتر است اما سطح متوسط ان هم قابل قبول است.
  - اثر بخشی در سطح متوسط هم به قدر کافی خوب است.

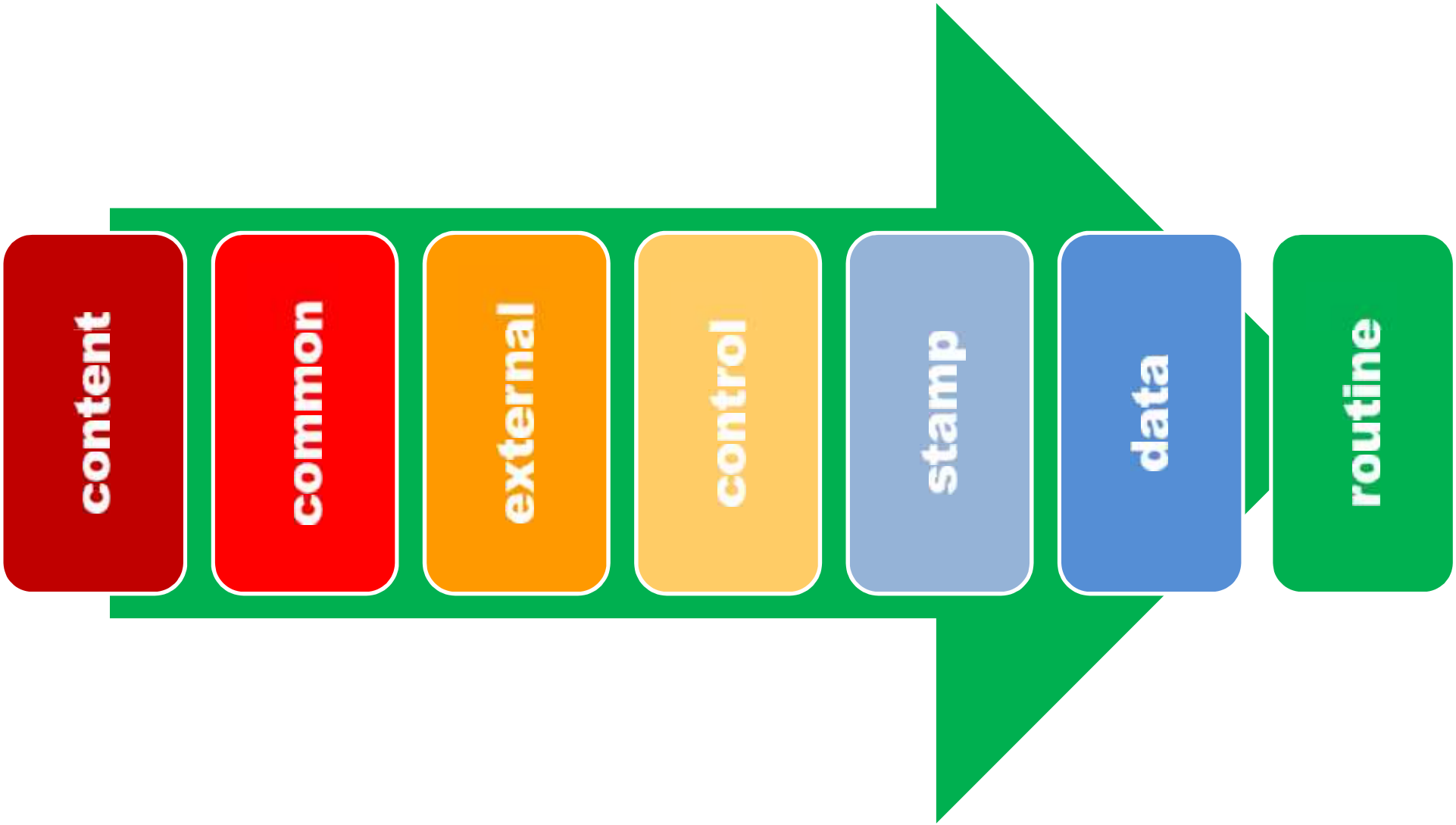
- **Coupling**
  - وابستکی بین ماژول ها را گویند.
  - هر چه ارتباط به دیگران بیشتر باشد Coupling بالاتر است.



پیوستگی

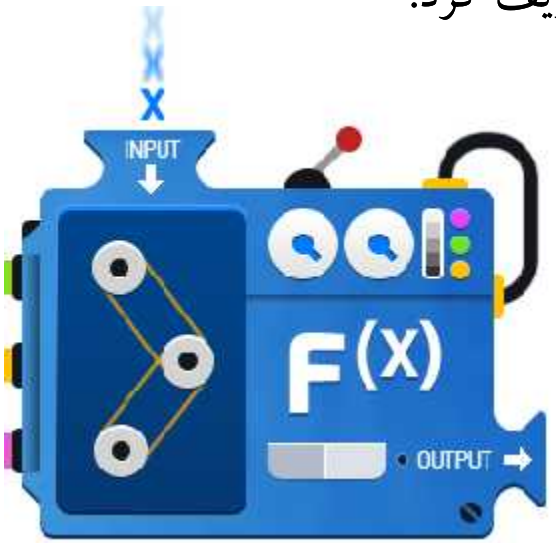

concidental
logical
temporal
Procedural
communica tional
layer
functional

وابستگی



تابع چیست؟

- در زبان پایتون نسخه سه همه چیز شی هست.
- در زبان پایتون توابع هم اشیا هستن و مانند تمام اشیای دیگر میتوان با انها کار کرد.
- توابع در پایتون جزو first-class citizens هستن. یعنی به صورت پویا میتوان انها را ساخت،نابود کرد،به عنوان ورودی به تابع دیگری داد، ان را return کرد و تمام ویژگی هایی که متغییرهای ساده دارند.
- این ویژگی در زبان هایی مثل Java و C# وجود ندارد.
- توابع را میتوان داخل module و class و یا function های دیگر تعریف کرد.

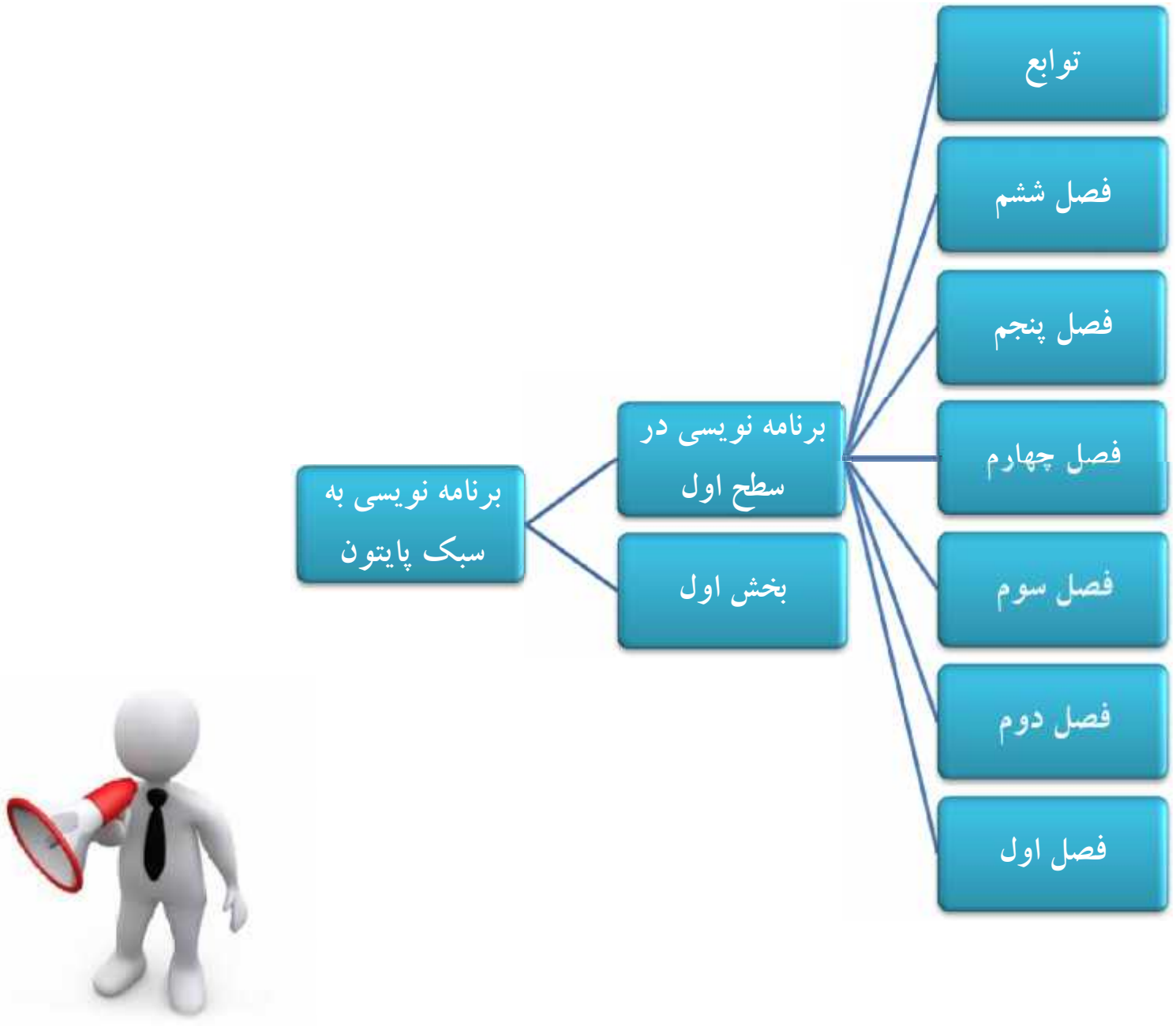


به نام پروردگار دانایی

برنامه نویسی به سبک پایتون

پدرام شاه صفی

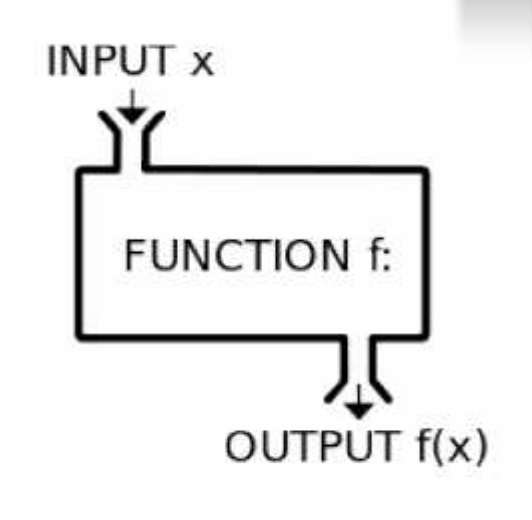
تابستان ۱۳۹۴


برنامه نویسی به سبک پایتون
برنامه نویسی در سطح اول
بخش اول
توابع
فصل ششم
فصل پنجم
فصل چهارم
فصل سوم
فصل دوم
فصل اول

کاری نکن!

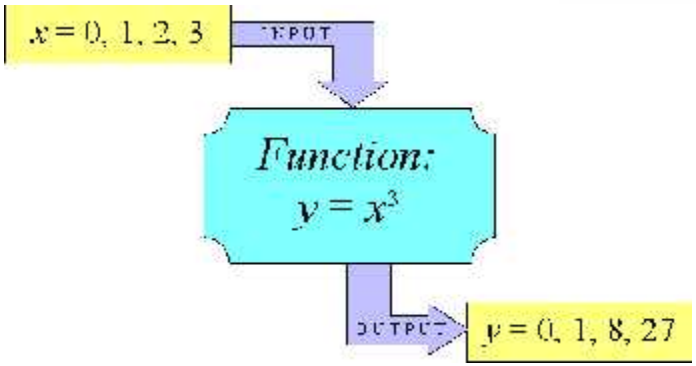
```
while True:
    pass  # Busy-wait for keyboard interrupt (Ctrl+C)
```

```
def initlog(*args):
    pass   # Remember to implement this!
```



تعریف تابع

```
def lowercase_separated_by_underscore():
    '''blah

    blah blah blah
    '''
    pass
    #function body
```

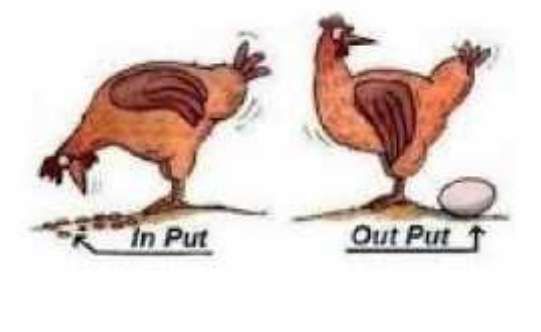


صدا زدن تابع

```
def func():
    print('run func')

func.__call__()
func()
func
```

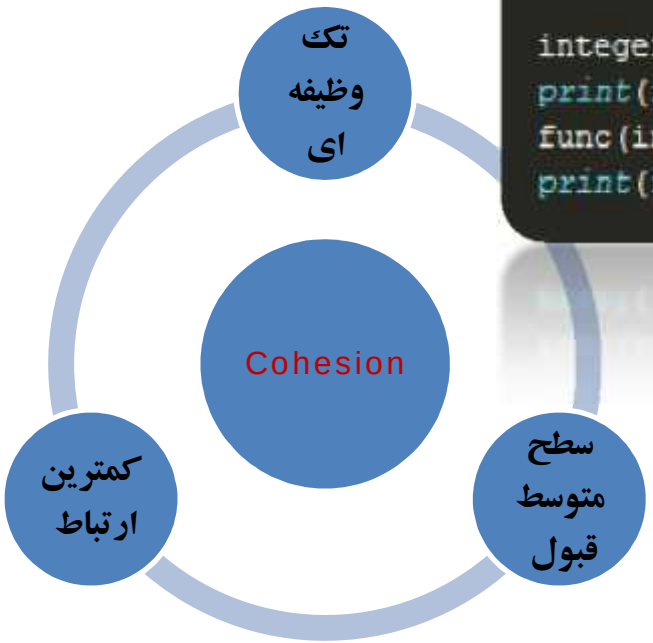
ورودی تابع

- انواع ورودی
  - منفرد
  - تاپل
  - دیکشنری



ارسال از طریق شی

```
def func(integer,List):
    print(id(integer),id(List))
    integer+=1
    List.append('new')
    print(id(integer),id(List))

integer,List=12,['python']
print(id(integer),id(List))
func(integer,List)
print(id(integer),id(List))
```

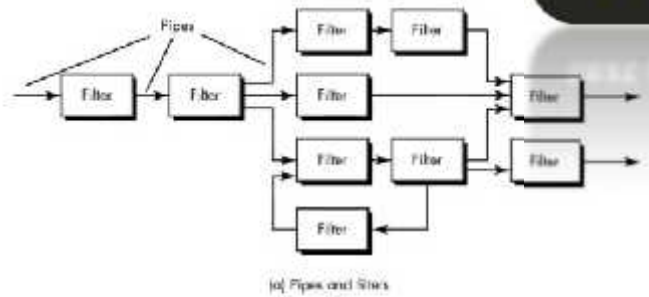


ورودی منفرد

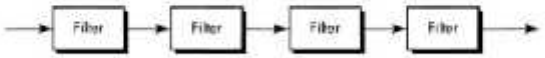

```python
import types

def foo(bar):
    if type(bar)==int:
        print('type=int()')
    elif type(bar)==str:
        print('type=str()')
    elif type(bar)==float:
        print('type=float()')
    elif type(bar)==tuple:
        print('type=tuple()')
    elif type(bar)==types.FunctionType:
        print('type=function()')
```

مثلا

```python
def test(x, *arguments, **keywords):
    print('simple: ', x)
    print("-" * 40)
    for arg in arguments:
        print(arg)
    print("-" * 40)
    keys = sorted(keywords.keys())
    for kw in keys:
        print(kw, ":", keywords[kw])

test('one','two','three',four=4,five=5)
```



(a) Pipes and filters

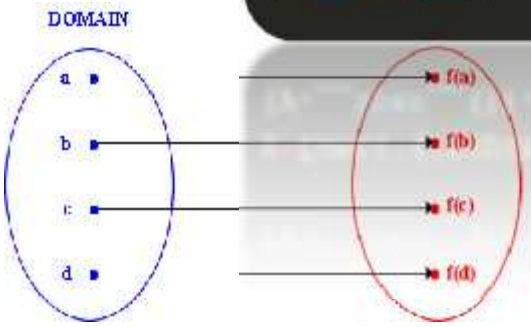


(b) Batch sequential

تجزیه

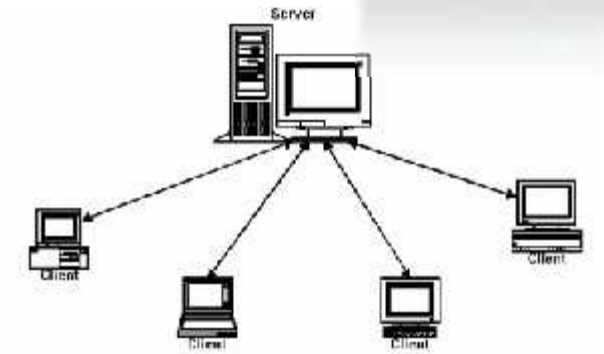
```
def func(x):
    print(type(x))
    print(x)
    return x
func(('tuple','or','list'))


def func(*x):
    print(type(x))
    print(x)
    return(x)

y=func(('tuple','or','list'))
(y.__iter__()).__next__()

y=func(*('tuple','or','list'))
(y.__iter__()).__next__()
```

```
def func(x,y,z):
    print('x=',x,'y=',y,'z=',z)

d=('one','two','three')
func(d)
func(*d)     #unpack tuple

d={'x':'one','y':'two','z':'three'}
func(d)
func(*d)
func(**d)    #unpack dict

d={'1':'one','2':'two','3':'three'}
func(**d)
```



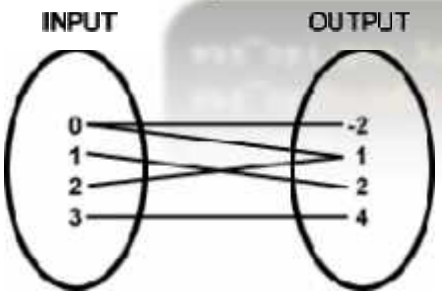
مقادیر پیش فرض

```
def ask_ok(prompt, retries=3, complaint='Yes or no, please!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            return None
        print(complaint)
```

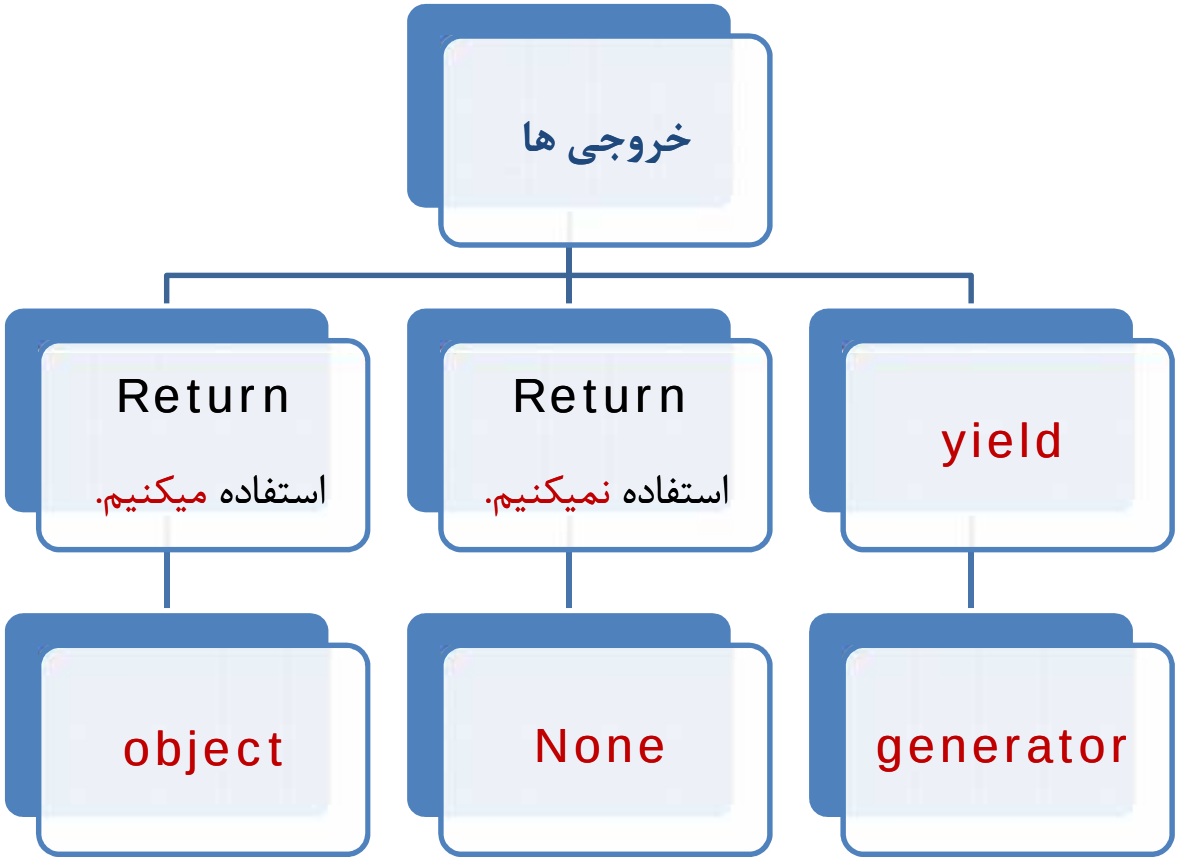


تغییر مقادیر پیش فرض

```
ask_ok('Do you really want to quit?')                #mandatory argument
ask_ok('OK to overwrite the file?', 2)               #optional arguments
ask_ok('OK to overwrite the file?', 2, 'y or n!')

ask_ok(prompt='Do you ?')                            # keyword argument
ask_ok(prompt='E you ?', retries=3 )                 # 2 keyword arguments
ask_ok(prompt='Are you ?',complaint='ops!')          # 2 keyword arguments
ask_ok(complaint='Yes or no, please!',prompt='message', retries=3 ) #all arguments

ask_ok()                                  #required argument missing
ask_ok(prompt='How you ?', 3 )            # non-keyword argument after a keyword argument
ask_ok('How you ?',prompt='Error'   )# duplicate value for the same argument
```



انواع خروجی




```
def foo():
    return 0
def bar():
    pass
def foo_bar():
    yield 5

ret=foo()
type(ret)
```

خروجی به روش مولد

```
def create_generator():
    for i in range(3):
        yield i*i

my_generator = create_generator() # create a generator
print(my_generator) # mygenerator is an object!
```

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]

for char in reverse('golf'):
    print(char)
```

خروجی چندتایی

```
def bar():
    yield 1
    yield 2
    yield 3

for i in bar():
    print(i)

def foo(x,y,z):
    return (x,y,z)

foo(1,2,3)
```

ورودی و خروجی مشخص

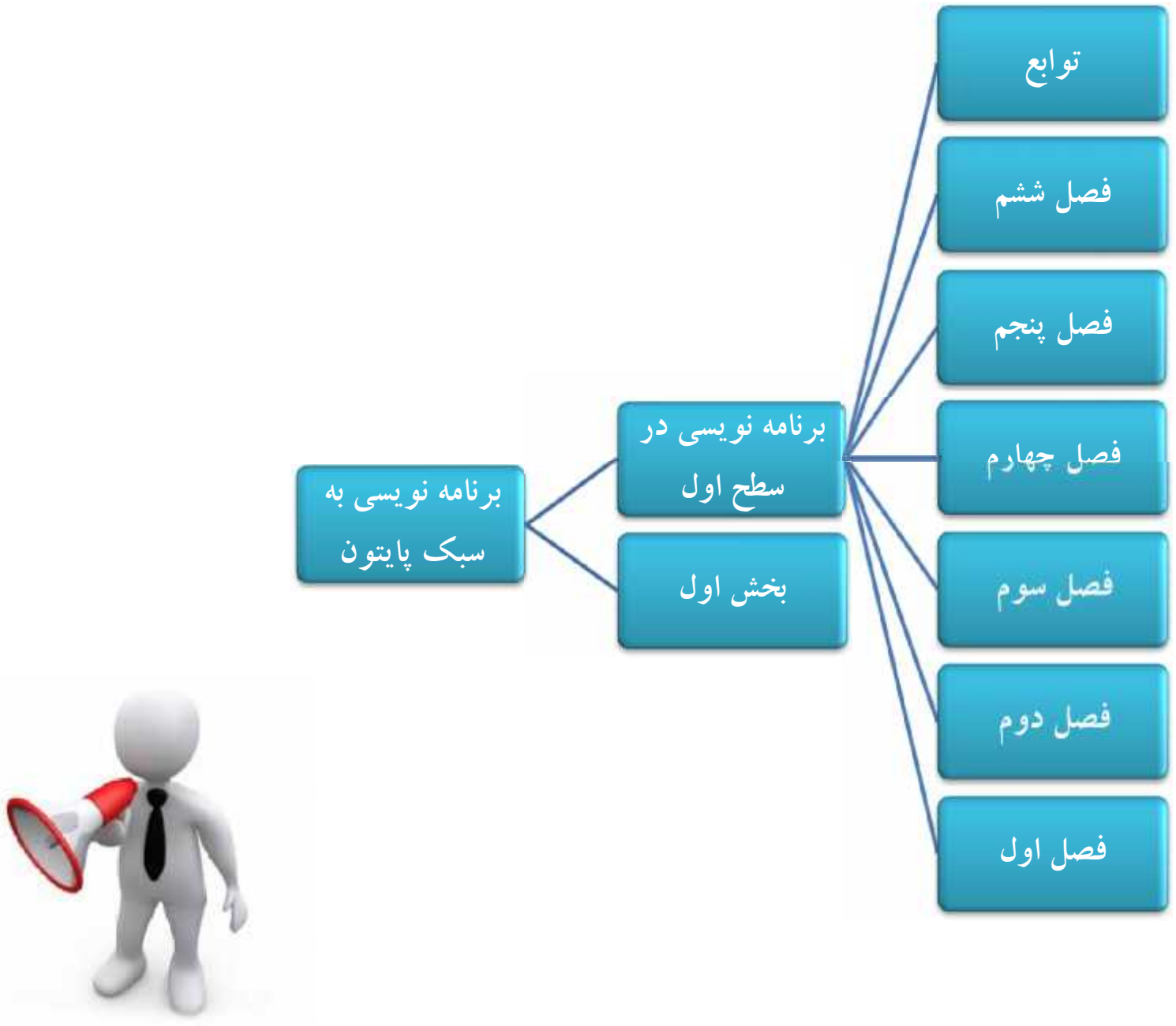
```
def f(ham: str, eggs: str = 'default') -> str:
    print("Annotations:", f.__annotations__)
    print("Arguments:", ham, eggs)
    return (ham + ' and ' + eggs)
```

به نام پروردگار دانایی

برنامه نویسی به سبک پایتون

پدرام شاه صفی

تابستان ۱۳۹۴


برنامه نویسی به سبک پایتون
برنامه نویسی در سطح اول
بخش اول
توابع
فصل ششم
فصل پنجم
فصل چهارم
فصل سوم
فصل دوم
فصل اول

وسعت دید

```
def x():
    print=12
    return print
print('wow')

name = "pd"
def f():
    global name
    name = "perl"
    #or
    name='python'
    print ("Within function", name)

print ("Outside function", name)
f()
print ("Outside function", name)

locals()
globals()
```

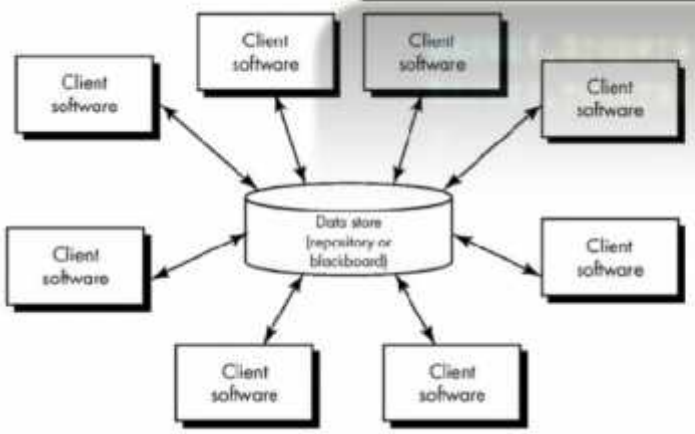
مقایسه

- globals
  - دیکشنری از current global symbol table یا current global namespace.اجزای ان scope of current module و built-ins هستن.
  - خروجی را بصورت (key, value)-pairs برمیگرداند. key در واقع یک رشته و نام symbol است.Value هم مقدار ان symbol است.
- locals
  - دیکشنری از current local symbol table یا current local namespace (scope of function).
- dir
  - بدون ورودی. یک لیست از اسامی current local scope یا current local namespace را برمیگرداند.
  - با ورودی. یک لیست از valid attribute ان ورودی برمیگرداند.
- vars
  - بدون ورودی. یک دیکشنری از current namespace.

مثلا

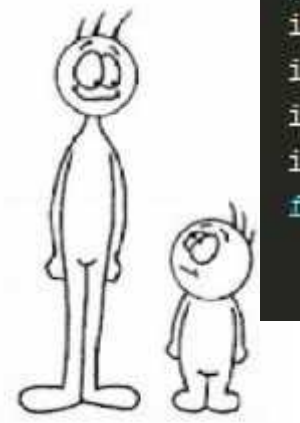
```
locals()
globals()
dir()
vars()

vars()==locals()
locals()==globals()
dir() == sorted( locals().keys() )

def a():
    l = 1
    print('locals() :',locals())
    print('globals() :',globals())
```

```
var=str('old_txt')
print(var)
def func():
    var='inside'
    globals()['var']='new_txt'
    print(var)
func()
print(var)
```



توابع ناشناس

```
def f(x):
    return x**2

g=lambda x:x**2

def mode(n):
    if n==1:
        return lambda x:x*x
    elif n==2:
        return lambda x:x**2
mode(1)
mode(1)(2)


x=(lambda x: x % 3 == 0, [2, 18, 9, 22, 17, 24, 8, 12, 27])
type(x)
iterator=filter(lambda x: x % 3 == 0, [2, 18, 9, 22, 17, 24, 8, 12, 27])
iterator
iterator()
iterator.__next__()
for i in iterator:
    print(i)
```

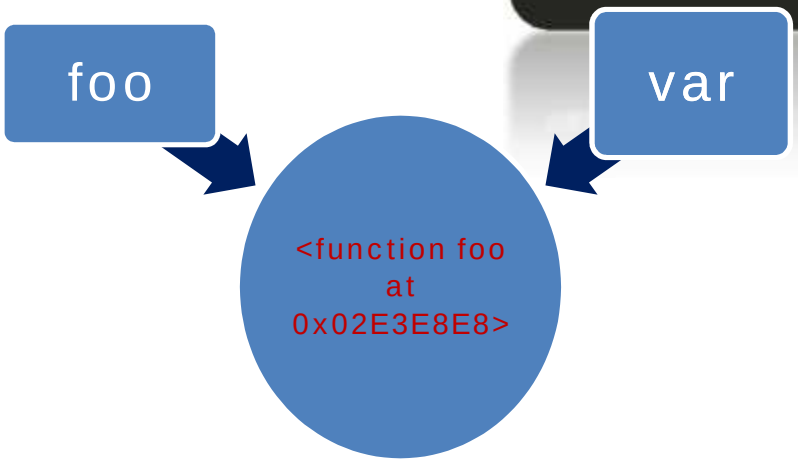


اجرای خودکار

```
def start_from_here():
    '''NAME

    time - This module provides various functions to manipulate time values.
DESCRIPTION

    There are two standard representations
    '''
    print('starting point ...')

x=int()
locals()
print(start_from_here.__name__)
print(start_from_here.__doc__)

if __name__=='__main__':
    start_from_here()
```

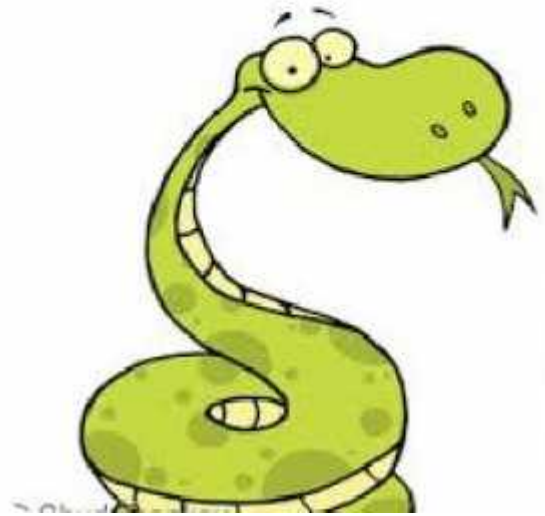
در پایتون همه چیز شی است!

```
def foo():
        print('kung fu')
foo      #functions are objects
var=foo  #var is foo
var()    #call var
del foo  #remove foo. function foo exists!
foo()    #call foo -> Error
var()    #call var
```



تابع داخلی

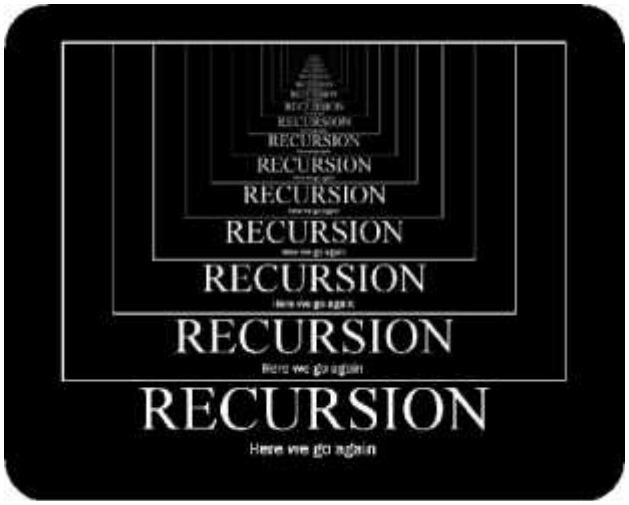
```
def outer():
        def inner():
                return('inner()')
        return ("outer()")

def outer():
        def inner():
                return('inner()')
        return inner

def outer():
        def inner():
                return('inner()')
        return inner()
outer
outer()
outer()()
```



صدا زدن یک تابع با تابع

```
def father():
    return "Father "

def call_fahter(f):
    return (f()+'called')

call_fahter.__call__(father)
```

```
def say_hello(name):
    return "hello"+name

def send_name(func):
    return func(" python!")

send_name(say_hello)
```



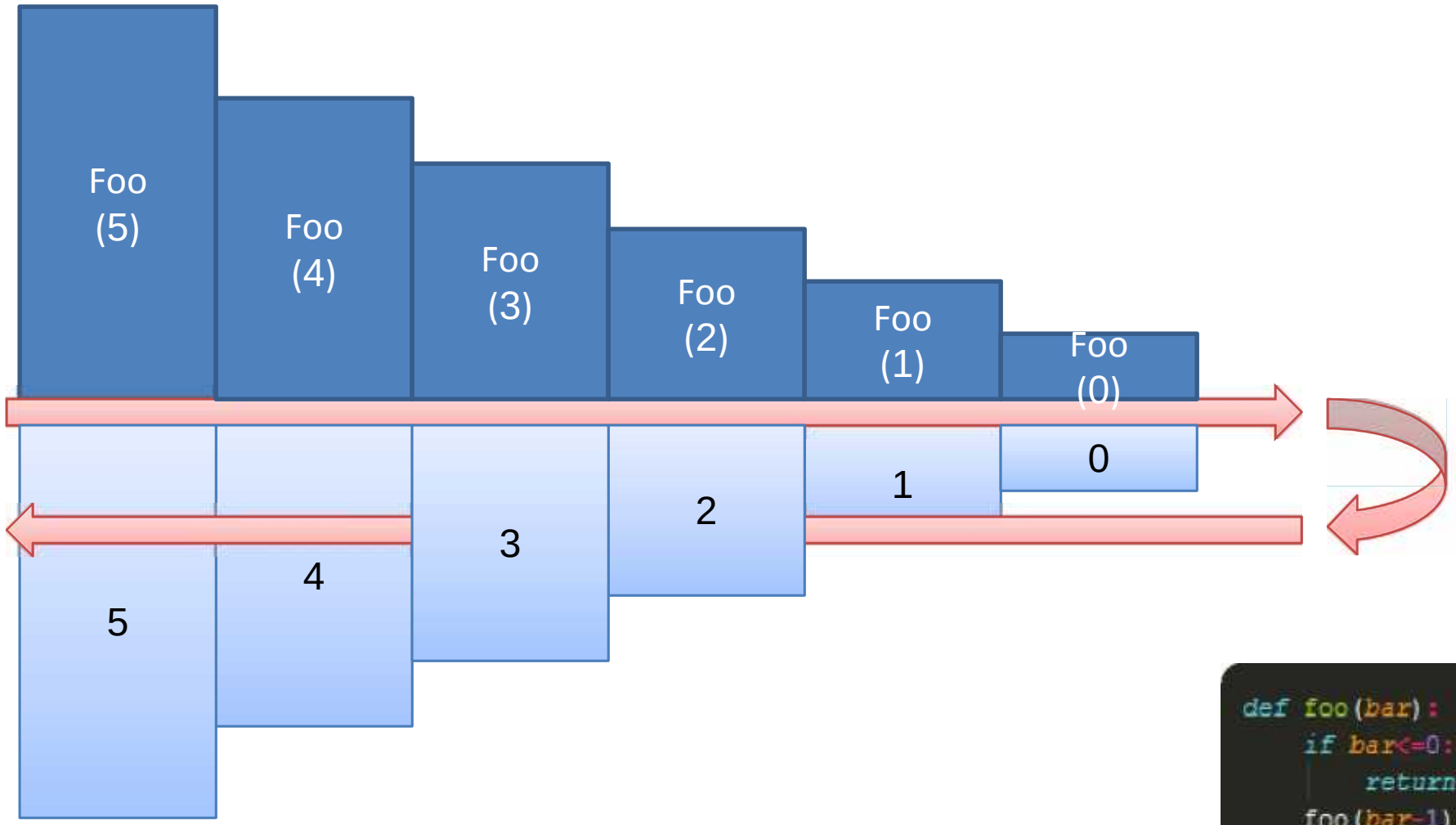
خروجی تابع، تابع

```
def main():
    def sub_func():
        print('subfunc')
        return sub_func
    return sub_func

main()
main()()
main()()()
main()()()()
```

مثلا

```
def my_func_1(x, **kwargs):
    if kwargs.get('plus_3'):
        return my_func_2(x, **kwargs) + 3
    return my_func_2(x, **kwargs)

def my_func_2(x, **kwargs):
    #Imagine that the function did more work
    if kwargs.get('square'):
        return x ** 2
    # If you decided to add cube as a parameter
    # you only need to change the code here:
    if kwargs.get('cube'):
        return x ** 3
    return x

my_func_1(5)
my_func_1(5, square=True)
my_func_1(5, plus_3=True, square=True)
my_func_1(5, cube=True)
```

توابع خود خوان

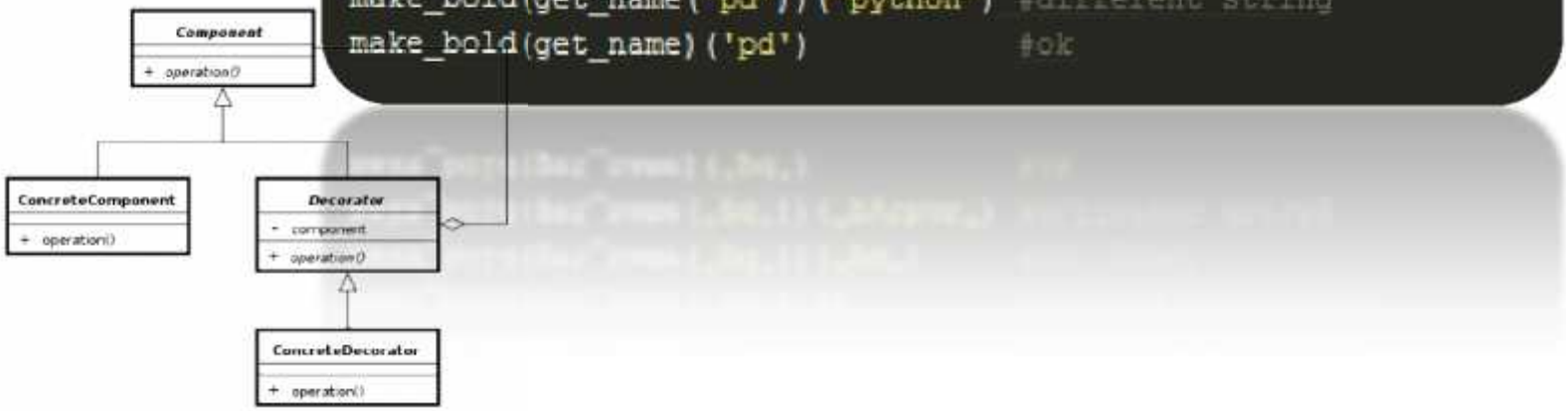



```
def foo(bar):
    if bar<=0:
        return 0
    foo(bar-1)
    print(bar)
```

اذینگر 😐 دستی

```python
def make_bold(func):
    def make_it(name):
        return "<b>"+name+"</b>"
    return make_it

def get_name(name):
    return name
```

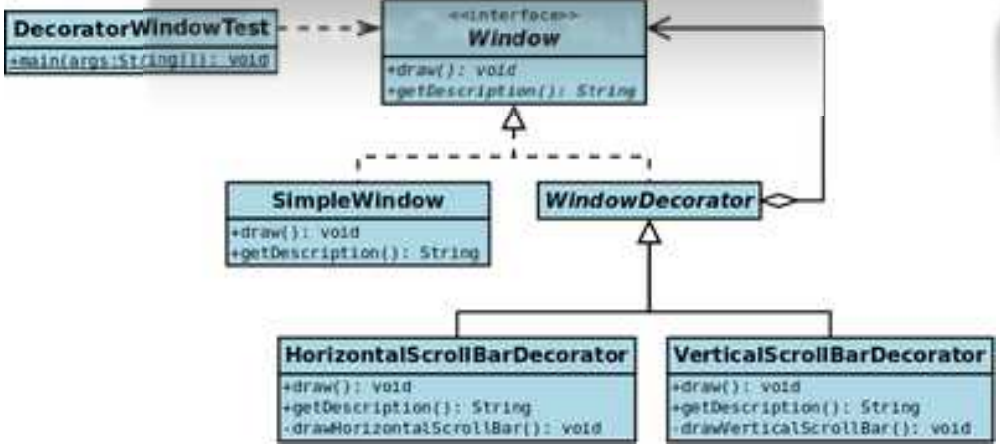
```python
get_name("pd")
make_bold(get_name)                  #make_bold.<locals>.make_it
make_bold(get_name("pd"))            #make_bold.<locals>.make_it
make_bold(get_name)()                #missing 1 required
make_bold(get_name("pd"))()          #missing 1 required
make_bold(get_name("pd"))("pd")      #not good!
make_bold(get_name('pd'))("python")  #different string
make_bold(get_name)('pd')            #ok
```



اذینگر😐

```python
def make_bold(func):
    def make_it(name):
        return "<b>"+name+"</b>"
    return make_it

def get_name(name):
    return name

make_bold(get_name)('pd')
```

```python
def make_bold(func):
    def make_it(name):
        return "<b>"+func(name)+"</b>"
    return make_it

@make_bold
def get_name(name):
    return name

get_name('pd')
```



مثلا

```python
import time
def time_dec(func):

  def wrapper(*arg):
      print(*arg)
      t = time.clock()
      res = func(*arg)
      print ( time.clock()-t)
      return res

  return wrapper


@time_dec
def foo(x,y):
    print('take the time!')
    return('foo output:',x*y)
```

تمرین

- با استفاده از yield *تابعی را بنویسید که محتوای یک فایل عظیم را به نحوه ی صحیح بخواند.*
- دستورات len()،max()،sum()،min() ، in و reversed() را بطور دستی پیاده سازی کنید.
- تابعی که رشته های palindromes را تشخیص دهد. رشته هایی اند که اگر از هر دو سمت چپ و راست شروع به خواندش کنید فرقی نکند .مثل radar .
- توابعی که رفتار صف و پشته را شبیه سازی کند.
- روش های تقسیم و حل، بازگشتی،حریصانه و پویا را بررسی کنید.
- مفاهیم کارایی، تحلیل و مرتبه ی الگوریتم ها را بررسی کنید.
- تابع Ackerman را پیاده سازی کنید.
- تابع فیبوناچی را بصورت بازگشتی بنویسید. برنامه ای برای بررسی عمکرد توابع بازگشتی بنویسید.
- برنامه ای که بررسی میکند که ایا تمام پرانتز های باز *شده*؛ بسته شده اند یا خیر .
- تابعی را بنویسید که با لیست یک بعدی زیر مانند لیستی دوبعدی رفتار کند.

```
Base Case. If m = 0
    return n +1.
N Zero Case. If m ≠ 0 and n = 0,
    return ackerman ( m-1 , 1 ).
N Non-Zero Case. If m ≠ 0 and n ≠ 0,
    return ackerman ( m-1 , ackerman ( m , n-1 ) ).
```

```
grid = ['top left',    'top middle',    'top right',
        'middle left', 'center',        'middle right',
        'bottom left', 'bottom middle', 'bottom right']

Print(0,0) # return 'top left'
Print(1,2) # return 'middle right'
```

```
Examples:

   []        OK    ][        NOT OK
   [][]      OK    ][][      NOT OK
   [[][]]    OK    []][[]    NOT OK
```

پاسخ

```python
def read_in_chunks(file_object, chunk_size=1024):
    """Lazy function (generator) to read a file piece by piece.
    Default chunk size: 1k."""
    while True:
        data = file_object.read(chunk_size)
        if not data:
            break
        yield data


f = open('really_big_file.dat')
for piece in read_in_chunks(f):
    process_data(piece)
```

پاسخ

```
def trace(f):
    f.indent = 0
    def g(x):
        print ('|  ' * f.indent + '|--', f.__name__, x)
        f.indent += 1
        value = f(x)
        print ('|  ' * f.indent + '|--', 'return', repr(value))
        f.indent -= 1
        return value
    return g

def fib(n):
    if n is 0 or n is 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

```
fib = trace(fib)
print (fib(2))
'''
|-- fib 2
|   |-- fib 1
|   |   |-- return 1
|   |-- fib 0
|   |   |-- return 1
|   |-- return 2
'''
```


RYKA
CBT