

Case Study: Python

Functional Programming

Dr. Nguyen Hua Phung

HCMC University of Technology, Viet Nam

08, 2020

- Immutable Data
- lambda function
- First-class functions
- High-order functions: map, filter, reduce
- Closure
- Decorator

- Syntax:

lambda (<param>(, <param>)*): <exp>

- Syntax:

lambda (<param>(, <param>)*): <exp>

- For example,

- Syntax:

lambda (<param>(, <param>)*): <exp>

- For example,

lambda a,b: a + b

- Syntax:

lambda (<param>(, <param>*)?): <exp>

- For example,

lambda a,b: a + b

(**lambda** a,b: a + b)(3,4)

- Syntax:

lambda (<param>(, <param>*)?): <exp>

- For example,

lambda a,b: a + b

(**lambda** a,b: a + b)(3,4) => 7

- Syntax:

lambda (<param>(, <param>*)?): <exp>

- For example,

lambda a,b: a + b

(**lambda** a,b: a + b)(3,4) => 7

x = **lambda** a,b: a + b

- Syntax:

lambda (<param>(, <param>*)?): <exp>

- For example,

lambda a,b: a + b

(**lambda** a,b: a + b)(3,4) => 7

x = **lambda** a,b: a + b

x(3,4)

- Syntax:

lambda (<param>(, <param>)*): <exp>

- For example,

lambda a,b: a + b

(**lambda** a,b: a + b)(3,4) => 7

x = **lambda** a,b: a + b

x(3,4) => 7

- Syntax:

lambda (<param>(, <param>*)?): <exp>

- For example,

lambda a,b: a + b

(**lambda** a,b: a + b)(3,4) => 7

x = **lambda** a,b: a + b

x(3,4) => 7

- Anonymous function

- Syntax:

lambda (<param>(, <param>*)?): <exp>

- For example,

lambda a,b: a + b

(**lambda** a,b: a + b)(3,4) => 7

x = **lambda** a,b: a + b

x(3,4) => 7

- Anonymous function
- Any number of parameters

- Syntax:

lambda (<param>(, <param>)*)?: <exp>

- For example,

lambda a,b: a + b

(**lambda** a,b: a + b)(3,4) => 7

x = **lambda** a,b: a + b

x(3,4) => 7

- Anonymous function
- Any number of parameters
- Body is just one expression

- Syntax:

lambda (<param>(, <param>)*):? : <exp>

- For example,

lambda a,b: a + b

(**lambda** a,b: a + b)(3,4) => 7

x = **lambda** a,b: a + b

x(3,4) => 7

- Anonymous function
- Any number of parameters
- Body is just one expression
- Used in high-order functions

First-class function

- A function is treated as any other value, i.e. it is
 - assigned to a variable

```
def foo(a,b): pass
```

```
x = foo
```

```
x(3,4)
```

First-class function

- A function is treated as any other value, i.e. it is
 - assigned to a variable

```
def foo(a,b): pass
```

```
x = foo
```

```
x(3,4)
```

- passed into another function as a parameter

```
def foo(f,x):
```

```
    return f(x)
```

```
foo(lambda a: a ** 2, 4)
```


First-class function

- A function is treated as any other value, i.e. it is
 - assigned to a variable

```
def foo(a,b): pass
```

```
x = foo
```

```
x(3,4)
```

- passed into another function as a parameter

```
def foo(f,x):
```

```
    return f(x)
```

```
foo(lambda a: a ** 2, 4) ==> 16
```

First-class function

- A function is treated as any other value, i.e. it is
 - assigned to a variable

```
def foo(a,b): pass  
x = foo  
x(3,4)
```

- passed into another function as a parameter

```
def foo(f,x):  
    return f(x)  
foo(lambda a: a ** 2, 4) ==> 16
```

- returned as a value

```
def f(x):  
    def g(y):  
        return x * y  
    return g  
m = f(3)  
m(4)
```

First-class function

- A function is treated as any other value, i.e. it is
 - assigned to a variable

```
def foo(a,b): pass  
x = foo  
x(3,4)
```

- passed into another function as a parameter

```
def foo(f,x):  
    return f(x)  
foo(lambda a: a ** 2, 4) ==> 16
```

- returned as a value

```
def f(x):  
    def g(y):  
        return x * y  
    return g  
m = f(3)  
m(4) ==> 12
```

- **map(<function>,<sequence>)**: apply **<function>** to each element of **<sequence>** and return an iterator

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator
cels = [36.5, 37, 37.5, 38, 39]

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]
```

```
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))
```

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]
```

```
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))
```

```
=> [97.7, 98.6, 99.5, 100.4, 102.2]
```

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]
```

```
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))
```

```
list(map(lambda x,y: x + y,[1,2,3],[4,5,6,7]))
```


- **map(<function>,<sequence>)**: apply **<function>** to each element of **<sequence>** and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]
```

```
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))
```

```
list(map(lambda x,y: x + y,[1,2,3],[4,5,6,7])) => [5,7,9]
```

High-order functions: map, filter, reduce

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]
```

```
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))
```

```
list(map(lambda x,y: x + y,[1,2,3],[4,5,6,7]))
```

- **filter(<function>,<sequence>)** return an iterator that contains elements in <sequence> for which <function> returns True

High-order functions: map, filter, reduce

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]
```

```
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))
```

```
list(map(lambda x,y: x + y,[1,2,3],[4,5,6,7]))
```

- **filter(<function>,<sequence>)** return an iterator that contains elements in <sequence> for which <function> returns True

```
list(map(lambda c: c % 2 == 1, [0,1,2,3,4,5]))
```

High-order functions: map, filter, reduce

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]
```

```
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))
```

```
list(map(lambda x,y: x + y,[1,2,3],[4,5,6,7]))
```

- **filter(<function>,<sequence>)** return an iterator that contains elements in <sequence> for which <function> returns True

```
list(map(lambda c: c % 2 == 1, [0,1,2,3,4,5])) => [1,3,5]
```

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]  
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))  
list(map(lambda x,y: x + y,[1,2,3],[4,5,6,7]))
```
- **filter(<function>,<sequence>)** return an iterator that contains elements in <sequence> for which <function> returns True

```
list(map(lambda c: c % 2 == 1, [0,1,2,3,4,5]))
```
- **reduce(<function>,<sequence>(<initial>)?)**: if <sequence> is $[s_1, s_2, s_3]$, **reduce** return
function(function(s_1, s_2), s_3) or
function(function(function(<initial>, s_1), s_2), s_3)

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]  
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))  
list(map(lambda x,y: x + y,[1,2,3],[4,5,6,7]))
```
- **filter(<function>,<sequence>)** return an iterator that contains elements in <sequence> for which <function> returns True

```
list(map(lambda c: c % 2 == 1, [0,1,2,3,4,5]))
```
- **reduce(<function>,<sequence>(<initial>)?)**: if <sequence> is $[s_1, s_2, s_3]$, **reduce** return
function(function(s_1, s_2), s_3) or
function(function(function(<initial>, s_1), s_2), s_3)
from functools import reduce

High-order functions: map, filter, reduce

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]
```

```
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))
```

```
list(map(lambda x,y: x + y,[1,2,3],[4,5,6,7]))
```

- **filter(<function>,<sequence>)** return an iterator that contains elements in <sequence> for which <function> returns True

```
list(map(lambda c: c % 2 == 1, [0,1,2,3,4,5]))
```

- **reduce(<function>,<sequence>(<initial>)?)**: if <sequence> is $[s_1, s_2, s_3]$, **reduce** return
function(function(s_1, s_2), s_3) or
function(function(function(<initial>, s_1), s_2), s_3)

```
from functools import reduce
```

```
reduce(lambda x,y: x+y,[1,2,3,4])
```

High-order functions: map, filter, reduce

- **map(<function>,<sequence>)**: apply <function> to each element of <sequence> and return an iterator

```
cels = [36.5, 37, 37.5, 38, 39]
```

```
fahr = list(map(lambda c: (float(9) / 5) * c + 32,cels))
```

```
list(map(lambda x,y: x + y,[1,2,3],[4,5,6,7]))
```

- **filter(<function>,<sequence>)** return an iterator that contains elements in <sequence> for which <function> returns True

```
list(map(lambda c: c % 2 == 1, [0,1,2,3,4,5]))
```

- **reduce(<function>,<sequence>(<initial>)?)**: if <sequence> is $[s_1, s_2, s_3]$, **reduce** return
function(function(s_1, s_2), s_3) or
function(function(function(<initial>, s_1), s_2), s_3)

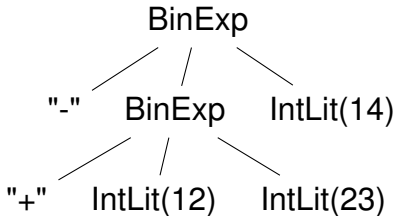
```
from functools import reduce
```

```
reduce(lambda x,y: x+y,[1,2,3,4]) ==> 10
```



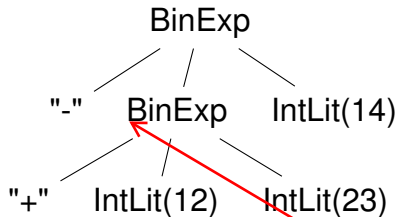
```
class Exp(ABC): pass
class IntLit(Exp): #val: int
class BinExp(Exp): #op:str, left:Exp, right:Exp
exp = [12, ("+", 23), ("−", 14)]
reduce(lambda acc, ele:
    BinExp(ele[0], acc, IntLit(ele[1])),
    exp[1:], IntLit(exp[0]))
```

```
class Exp(ABC): pass  
class IntLit(Exp): #val: int  
class BinExp(Exp): #op:str, left:Exp, right:Exp  
exp = [12, ("+", 23), ("-", 14)]  
reduce(lambda acc, ele:  
    BinExp(ele[0], acc, IntLit(ele[1])),  
    exp[1:], IntLit(exp[0]))
```



Example

```
class Exp(ABC): pass
class IntLit(Exp): #val: int
class BinExp(Exp): #op:str, left:Exp, right:Exp
exp = [12, ("+", 23), ("-", 14)]
reduce(lambda acc, ele:
    BinExp(ele[0], acc, IntLit(ele[1])),
    exp[1:], IntLit(exp[0]))
```



```
acc = IntLit(exp[0])
for ele in exp[1:]:
    acc = BinExp(ele[0],
                  acc,
                  IntLit(exp[1]))
```

đổi vị trí BinExp với IntLit(14) thì sao?
=> ko sao cả, tùy vào yêu cầu bài (ko liên
quan đến kết hợp gì)
~ nhưng nên để như vậy nếu kết hợp trái

- **Closure** is a function object together with an environment.

```
def power(y):  
    def inner(x):  
        return x ** y  
    return inner  
square= power(2)  
square(5)
```

- **Closure** is a function object together with an environment.

```
def power(y):  
    def inner(x):  
        return x ** y  
    return inner  
square= power(2)  
square(5) => 25
```

- **Decorator** allows to modify the behavior of function or class without permanently modifying it.

- **Decorator** allows to modify the behavior of function or class without permanently modifying it.

```
def foo(x,y):  
    return x*y  
print(foo(3,4))
```

- **Decorator** allows to modify the behavior of function or class without permanently modifying it.

```
def foo(x,y):  
    return x*y  
print(foo(3,4))    => 12
```


- **Decorator** allows to modify the behavior of function or class without permanently modifying it.

```
@log_decorator  
def foo(x,y):  
    return x*y  
print(foo(3,4))
```

- **Decorator** allows to modify the behavior of function or class without permanently modifying it.

```
@log_decorator
def foo(x, y):
    return x*y
print(foo(3,4))
    => foo is running
    => 12
```

- **Decorator** allows to modify the behavior of function or class without permanently modifying it.

```
@log_decorator  
def foo(x,y):  
    return x*y  
print(foo(3,4))
```

- How?

```
def log_decorator(func):  
    def inner(*arg):  
        print(func.__name__+"_is_running")  
        return func(*arg)  
    return inner
```

- **Decorator** allows to modify the behavior of function or class without permanently modifying it.

```
@log_decorator  
def foo(x,y):  
    return x*y  
print(foo(3,4))
```

```
def foo(x,y):  
    return x*y  
foo = log_decorator(foo)  
print(foo(3,4))
```

- How?

```
def log_decorator(func):  
    def inner(*arg):  
        print(func.__name__+"_is_running")  
        return func(*arg)  
    return inner
```

- [1] Python Tutorial, <http://w3schools.com/python>, 10 08 2020.
- [2] Python Programming Language, <https://www.geeksforgeeks.org/python-programming-language/>, 10 08 2020.
- [3] Python Tutorial, <https://www.tutorialspoint.com/python>, 10 08 2020.
- [4] Introduction to Python 3, <https://realpython.com/python-introduction/>, 10 08 2020.