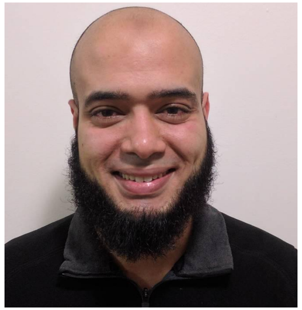
Python Programming
Lambda Function

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

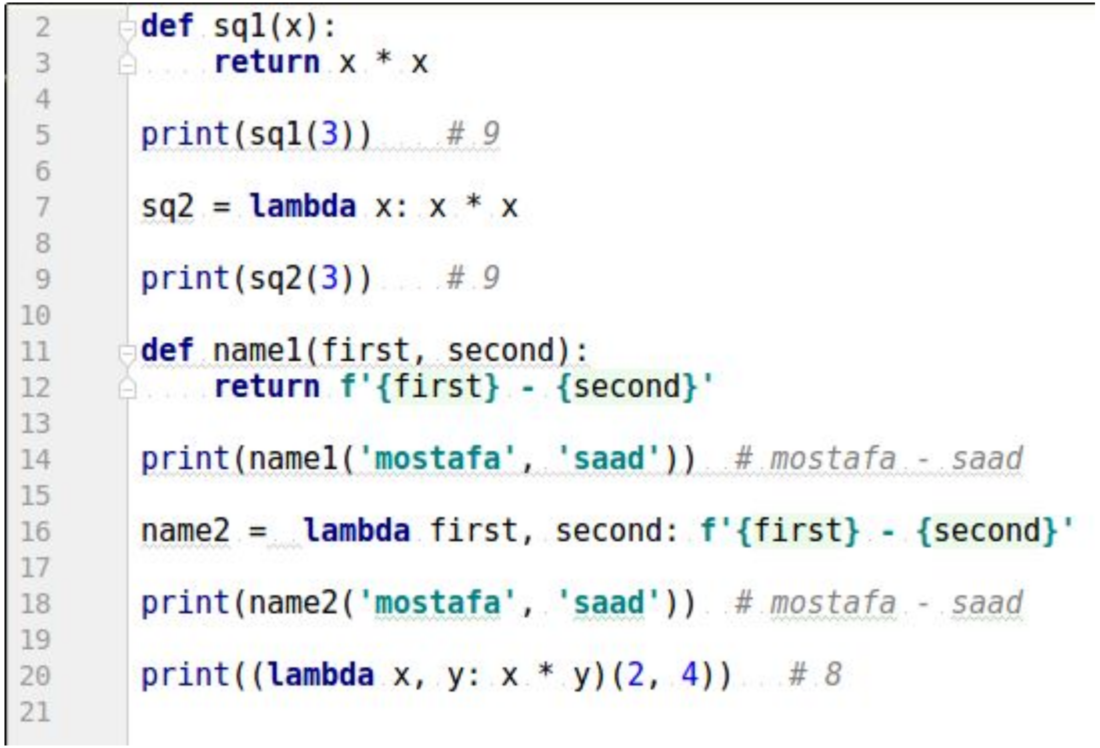


Lambda function

- An **anonymous** function is a function without a **name**.
  - We call them **lambda** function (or expression) also
- Pros:
  - Shorthand notation that makes some code use cases nicer sometime
- Cons:
  - No good stacktrace
    - You better use it for simple things that probably won’t cause problems
  - It doesn’t work well with static type checker (like `mypy` or `pyre`) [future]
    - In python 3: we can indicate expected type
  - Limited for only a single **expression**, NO statements

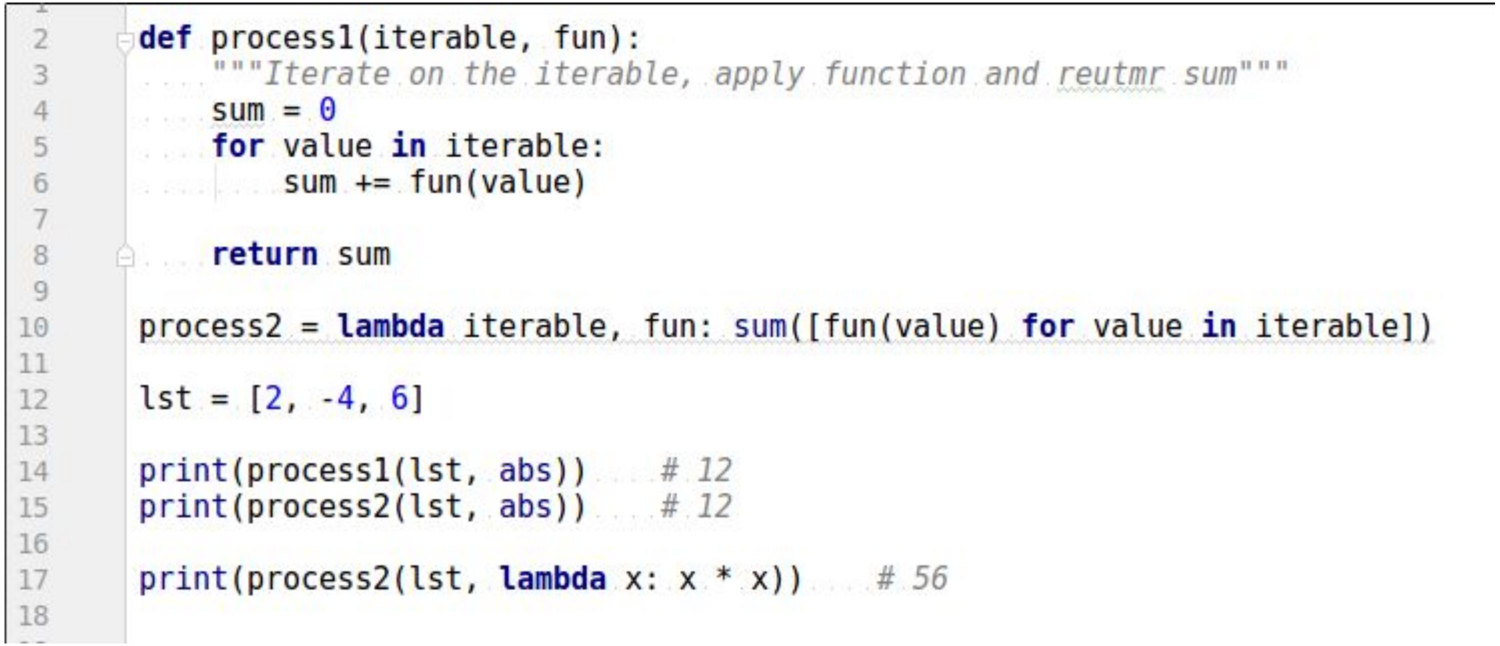
From normal function to lambda

- Compare the normal function with lambda function to observe the syntax changes



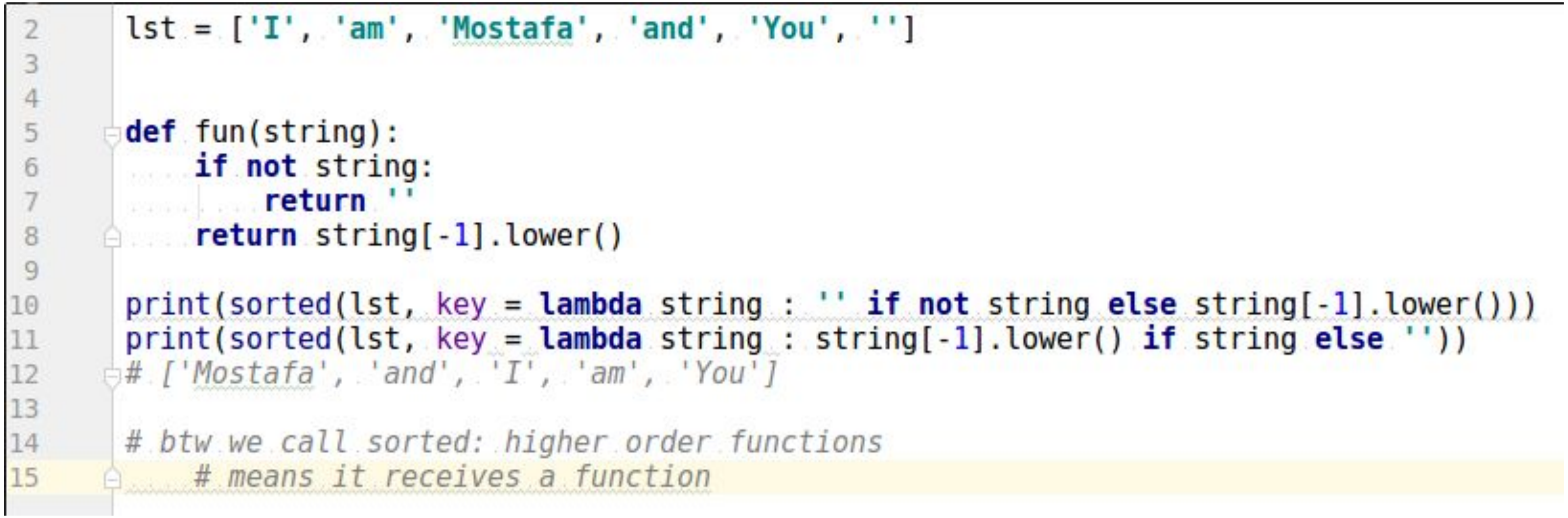
```
def sq1(x):
    return x * x

print(sq1(3))    # 9

sq2 = lambda x: x * x

print(sq2(3))    # 9

def name1(first, second):
    return f'{first} - {second}'

print(name1('mostafa', 'saad'))  # mostafa - saad

name2 =  lambda first, second: f'{first} - {second}'

print(name2('mostafa', 'saad'))  # mostafa - saad

print((lambda x, y: x * y)(2, 4))   # 8
```

From normal function to lambda



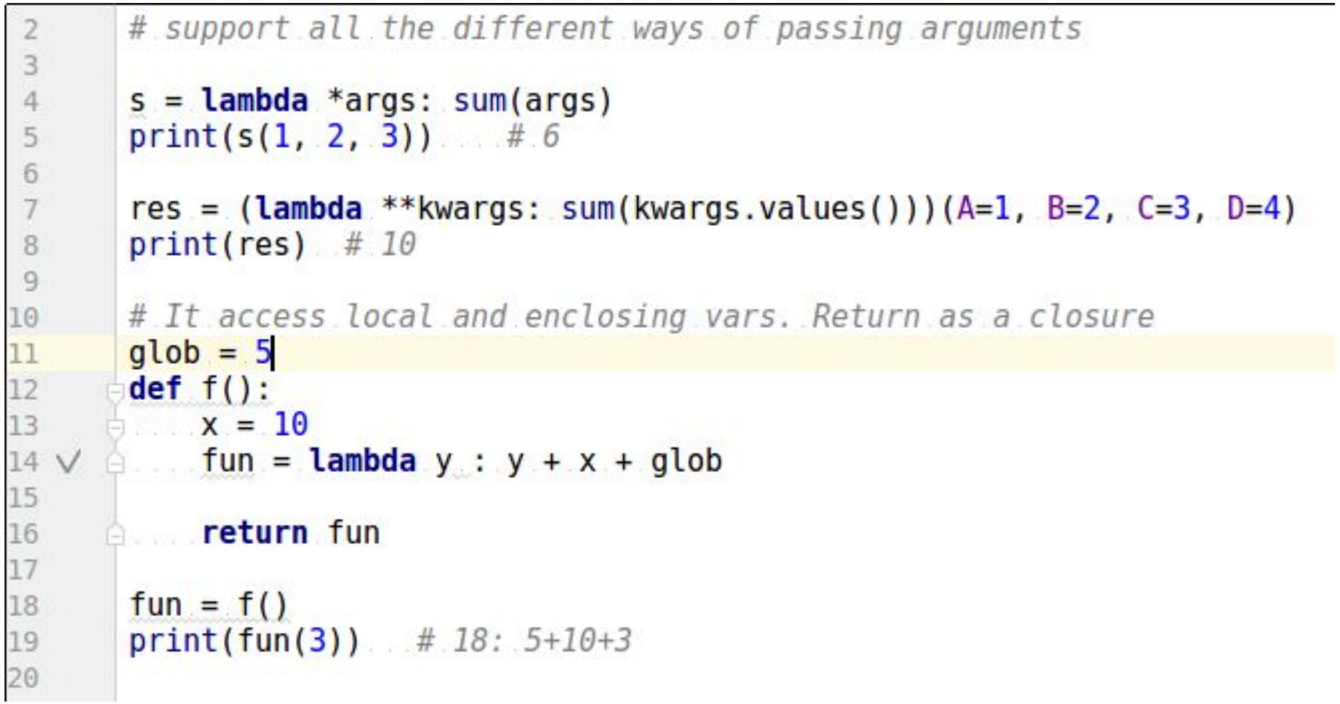
```python
def process1(iterable, fun):
    """Iterate on the iterable, apply function and reutmr sum"""
    sum = 0
    for value in iterable:
        sum += fun(value)

    return sum

process2 = lambda iterable, fun: sum([fun(value) for value in iterable])

lst = [2, -4, 6]

print(process1(lst, abs))    # 12
print(process2(lst, abs))    # 12

print(process2(lst, lambda x: x * x))    # 56

```

With Higher Order Functions



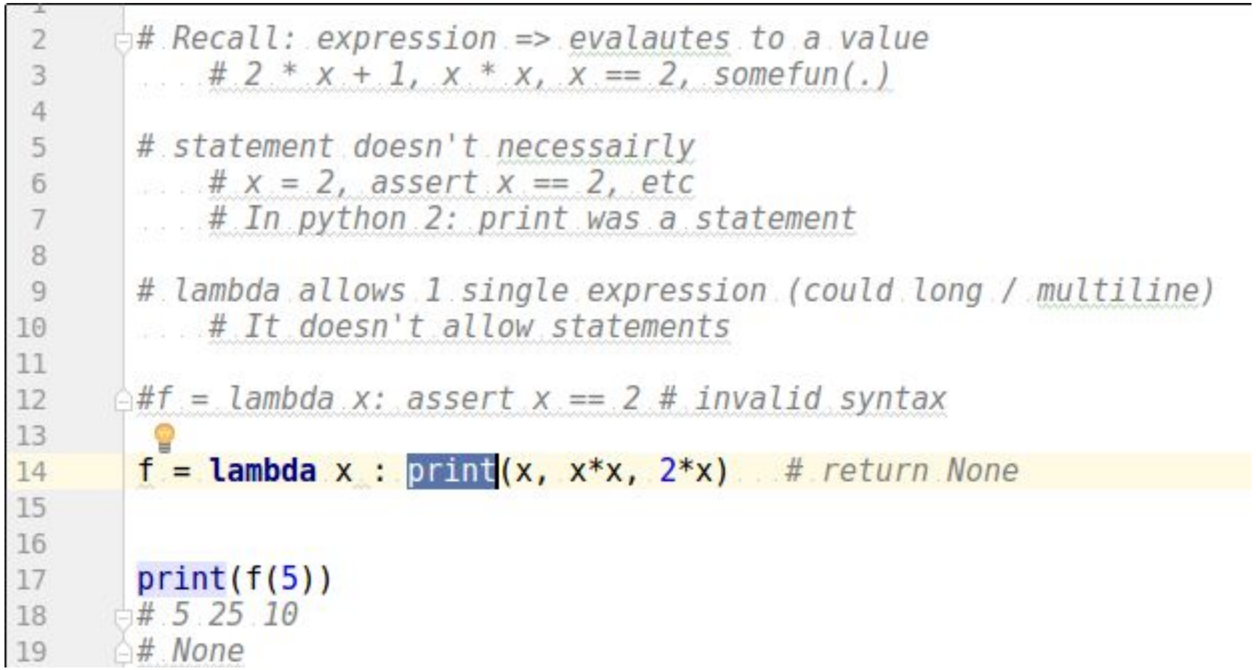
```python
lst = ['I', 'am', 'Mostafa', 'and', 'You', '']


def fun(string):
    if not string:
        return ''
    return string[-1].lower()

print(sorted(lst, key = lambda string : '' if not string else string[-1].lower()))
print(sorted(lst, key = lambda string : string[-1].lower() if string else ''))
# ['Mostafa', 'and', 'I', 'am', 'You']

# btw we call sorted: higher order functions
    # means it receives a function
```

Like normal functions



```
# support all the different ways of passing arguments

s = lambda *args: sum(args)
print(s(1, 2, 3))    # 6

res = (lambda **kwargs: sum(kwargs.values()))(A=1, B=2, C=3, D=4)
print(res)  # 10

# It access local and enclosing vars. Return as a closure
glob = 5
def f():
    x = 10
    fun = lambda y : y + x + glob

    return fun

fun = f()
print(fun(3))   # 18: 5+10+3
```

Single Expression ONLY



```
# Recall: expression => evalautes to a value
    # 2 * x + 1, x * x, x == 2, somefun(.)

# statement doesn't necessairly
    # x = 2, assert x == 2, etc
    # In python 2: print was a statement

# lambda allows 1 single expression (could long / multiline)
    # It doesn't allow statements

#f = lambda x: assert x == 2 # invalid syntax

f = lambda x : print(x, x*x, 2*x)   # return None


print(f(5))
# 5 25 10
# None
```

Finally

- There are debates around lambda and their usage / issues
- I like this quote: “lambda functions are perfectly Pythonic if there is nothing more Pythonic available”
- Replacements include list comprehension, map, filter, reduce [future]
  - We already knows the first 2
- Future reading

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."