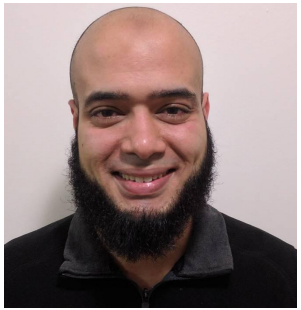
Python Programming

Closure

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)



Enclosing scope

- 2 useful use cases for nested functions that access enclosing scope
- 1) DRY (Don’t repeat yourself)
  - If there is a logic that repeats a lot, just move to inner function
  - E.g. you do some preprocessing for something, then write a line to file
    - You need to access some of the available vars in the enclosing scope
- 2) Closure
  - The outer function return the inner function (not calling result)
    - inner not inner(10, 20)
  - The returned function will REMEMBER the used enclosing variables EVEN after the return!
    - It captures **variables** NOT values
    - Used with Python Decorators (later)

Closure

- The return of inner is a closure that will keep binding with enclosing variables x and y

```
3   def outer(x):
4       y = 20
5       print(id(y))
6
7       def inner(f):
8           print(id(y))
9           return x + y + f
10
11      return inner
12
13
14  if __name__ == '__main__':
15      f = outer(10)
16      print(f(30))  # 60: 10 + 20 + 30
17      print(f(40))  # 70: 10 + 20 + 40
18
19      print(outer(100)(5))    # 125
20
```

Example

```
def init():
    class CustomersDataBase:
        def load_database(self):
            nonlocal users_ids
            users_ids += [3, 4]

        def add_id(self, id):
            if id not in users_ids:
                print(f'Adding {id}')
                # doesn't need nonlocal
                users_ids.append(id)
                print(users_ids)
            else:
                print(f'{id} is already there')

    users_ids = [1, 2]
    db = CustomersDataBase()
    db.load_database()

    return db.add_id
```

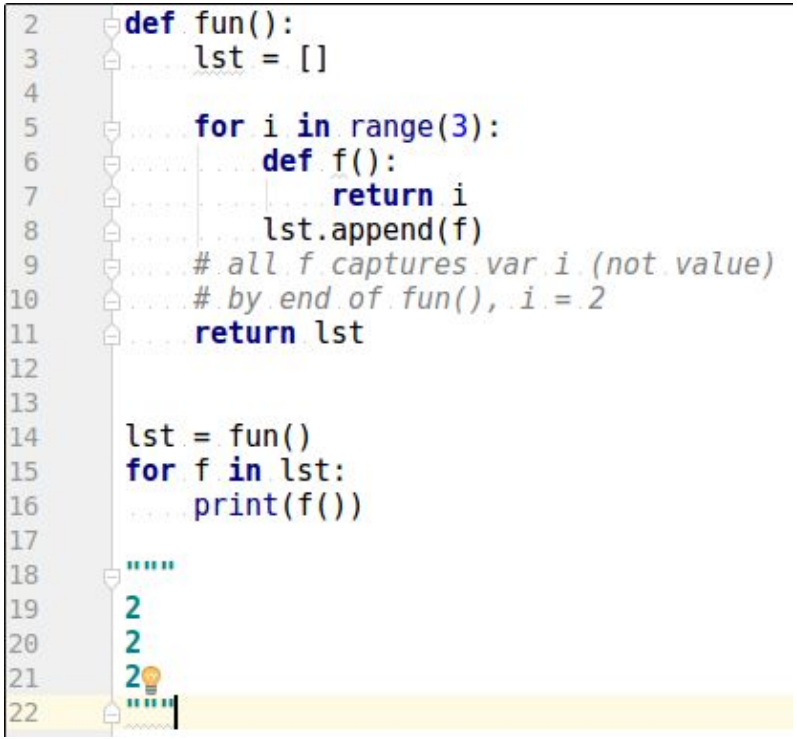
```
def go1(adder):
    adder(4)
    adder(5)

def go2(adder):
    adder(6)

if __name__ == '__main__':
    id_adder = init()
    go1(id_adder)
    go2(id_adder)
"""
4 is already there
Adding 5
[1, 2, 3, 4, 5]
Adding 6
[1, 2, 3, 4, 5, 6]
"""
```

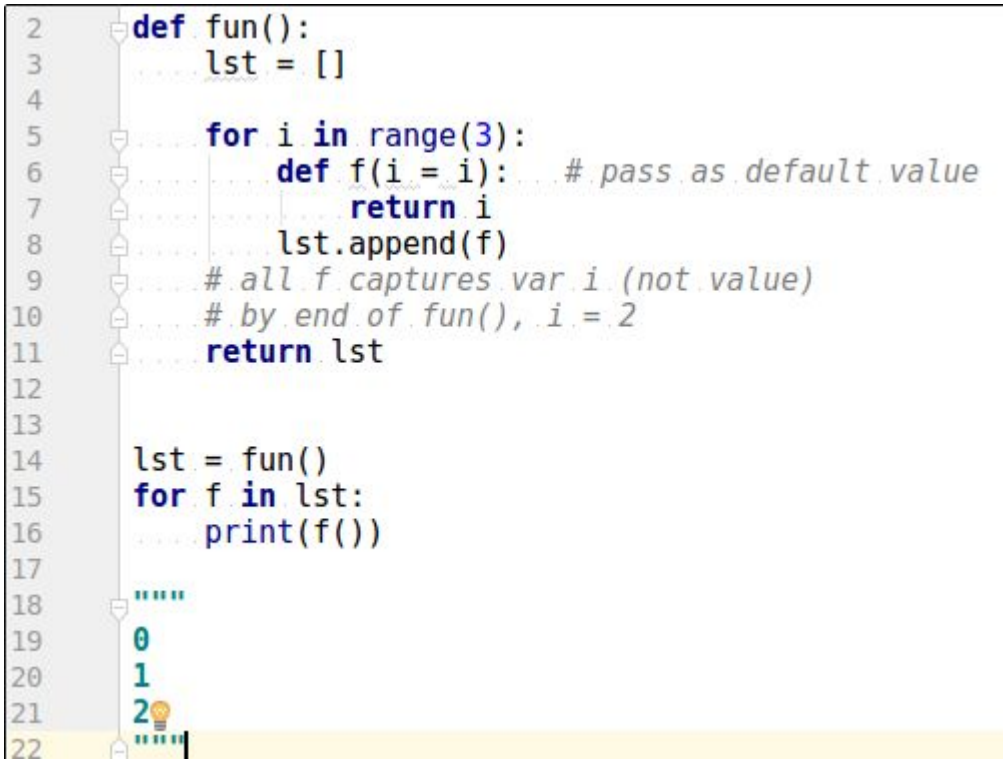
Be Careful

- Variable i is captured in f(), but although each capture has i with specific value, after we return, the final i value is used. Closures capture variables NOT values



Workaround

- Add parameter with default value



"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."