# CS 1520: Recitation 5

More Python

Fall 2018

# Python Functions

- First Class Objects
    - Can be referenced by a variable, added in a list, passed as argument to another function etc
- Can be defined inside another function
- Can return references to another function

https://hackernoon.com/decorators-in-python-8fd0dce93c08

# Exercise – Function as argument to a function

Give output of the following:

```
def say_hello(say_hi_func):
        print("Hello")

        say_hi_func()

def say_hi():
        print("Hi")

say_hello(say_hi)
```

Output:

```
Hello
Hi
```

https://hackernoon.com/decorators-in-python-8fd0dce93c08

# Exercise – Function inside another function

- Give output of the following:

```
def say_hello():
    print("Hello")

    def say_hi():
        print("Hi")

    say_hi()

say_hello()
say_hi()
```

Output:

```
Hello
Hi
Traceback (most recent call last):
  File "test.py", line 20, in <module>
    say_hi() # Gives error
NameError: name 'say_hi' is not
defined
```

https://hackernoon.com/decorators-in-python-8fd0dce93c08

# Exercise – Function returned by a function

- Give output of the following:

```
def say_hello():
    print("Hello")

    def say_hi():
        print("Hi")

    return say_hi

say_hi_func = say_hello()
say_hi_func()
```

Output:

```
Hello
Hi
```

https://hackernoon.com/decorators-in-python-8fd0dce93c08

# Decorators

- Acts as a wrapper to the original function

- Advantages:
  - Avoids code duplication
  - Does not clutter original code with additional logic

https://www.youtube.com/watch?v=nYDKH9fvlBY

# Example

- Finding the execution time of all the functions in a given program
- See decorator.py

https://www.youtube.com/watch?v=nYDKH9fvlBY

# Exercise:

```python
def plus(func):
   def wrapper(x,y):
      return func(x+10, y+10)
   return wrapper

def multi(func):
   def wrapper(x,y):
      return func(x*2, y)
   return wrapper

@plus
@multi
def add(x,y):
   return x+y

print (add(10,10))
```

Output:

```
60
```

# Generators

- Simple Way of creating iterators

- Example

```
def square_numbers(nums):
    for i in nums:
        yield (i*i)

my_nums = square_numbers([1,2,3,4,5])
print (my_nums)
for num in my_nums:
    print (num)
```

Output:

```
<generator object square_numbers at
0x0000021306B124F8>
1
4
9
16
25
```

https://www.youtube.com/watch?v=bD05uGo_sVI

# Exercise

- Give Output of the following:

```
my_nums = (x*x for x in [1,2,3,4,5])
print (my_nums)
for num in my_nums:
    print (num)
```

Output:

```
<generator object square_numbers at
0x0000021306B124F8>
1
4
9
16
25
```

https://www.youtube.com/watch?v=bD05uGo_sVI

# Python try, except and finally

- If an error is encountered, code execution in a try block is stopped and transferred down to the except block

- The code in the finally block will be executed regardless of whether an exception occurs.

https://www.pythonforbeginners.com/error-handling/python-try-and-except

# Exercise

- Give output of the following code snippet:

```
(m, n, x, y) = (2, 4, 10000, 0)
try:
    print (m/n)
    print (x/y)
except ZeroDivisionError:
  print("division by zero!")
else:
  print("Total Success")
finally:
  print("executing finally clause")
```

- Output:

```
0.5
division by zero!
executing finally clause
```

https://docs.python.org/3/tutorial/errors.html

# Python Classes

- A class is a code template for creating objects
  - A blueprint
  - Uses the keyword *class*

- A variable of the given class type is called it's instance/object.
  - Instance = class_name(arguments)

https://en.wikipedia.org/wiki/Class_(computer_programming)

# Exercise

Give output of the following:

```
class Employee:
    def __init__(self, first, last):
        self.first = first
        self.last = last
    def fullname(self):
        return self.first+" "+self.last

emp_1 = Employee('Albert', 'Einstein')

print(emp_1.fullname())
print(Employee.fullname(emp_1))
```

- Output:

```
Albert Einstein
Albert Einstein
```

https://www.youtube.com/watch?v=ZDa-Z5JzLYM

# Exercise

Give output of the following:

```
class Employee:
   def __init__(self, first, last):
      self.first = first
      self.last = last
   def fullname():
      return self.first+" "+self.last

emp_1 = Employee('Albert', 'Einstein')

print(emp_1.fullname())
print(Employee.fullname(emp_1))
```

- Output:

```
Traceback (most recent call last):
  File "test.py", line 19, in <module>
    print(emp_1.fullname())
TypeError: fullname() takes 0
positional arguments but 1 was given
```

https://www.youtube.com/watch?v=ZDa-Z5JzLYM

# Class Variable - Exercise

- Give output of the following:

```
class Employee:
    classamount = 100
    def __init__(self, first, last):
        self.first = first
        self.last = last

emp_1 = Employee('Albert', 'Einstein')
emp_2 = Employee('Isaac', 'Newton')
emp_1.classamount = 200
print(emp_1.classamount)
print(emp_2.classamount)
print(Employee.classamount)
```

Output

```
200
100
100
```

https://www.youtube.com/watch?v=BJ-VvGyQxho

# Class Method and Static Method

- Class Method
  - Bound to the class, not to the object.
  - Can access/modify the state of the class
  - Takes a class parameter as its first argument
  - Uses the built-in @classmethod decorator

- Static Method
  - Also bound to the class, not to the object
  - Cannot access/modify state of the class
  - Needs no specific parameters
  - Uses @staticmethod decorator

https://www.geeksforgeeks.org/class-method-vs-static-method-python/

# Example

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    @classmethod
    def fromBirthYear(cls, name, year):
        return cls(name, date.today().year - year)
    @staticmethod
    def isAdult(age):
        return age > 18

person1 = Person('Harry', 40)
person2 = Person.fromBirthYear(‘Harry', 1978)
print (person1.age)
print (person2.age)
print (Person.isAdult(22))
```

Output:

```
40
40
True
```

https://www.geeksforgeeks.org/class-method-vs-static-method-python/

# Inheritance

- Basic syntax of a subclass definition looks like:

```
class DerivedClassName(BaseClassName):
    pass
```

https://www.python-course.eu/python3_inheritance.php

# Exercise

```
class Person:
    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last
    def Name(self):
        return self.firstname + " " + self.lastname
class Employee(Person):
    def __init__(self, first, last, staffnum):
        super().__init__(first, last)
        self.staffnumber = staffnum
    def GetEmployee(self):
        return self.Name() + ", " +  self.staffnumber

x = Person("Marge", "Simpson")
y = Employee("Homer", "Simpson", "1007")
print(x.Name())
print(y.GetEmployee())
```

Output:

```
Marge Simpson
Homer Simpson, 1007
```

https://www.python-course.eu/python3_inheritance.php

# Overriding - Exercise

Give output of the following:

```
class Parent:
  def myMethod(self):
    print ('Calling parent method')

class Child(Parent):
  def myMethod(self):
    print ('Calling child method')

c = Child()
c.myMethod()
```

Output:

```
Calling child method
```

https://www.tutorialspoint.com/python/python_classes_objects.htm

# Exercise

```
class Parent:
  parentAttr = 100
  def __init__(self):
    print ("Calling parent constructor")
  def parentMethod(self):
    print ('Calling parent method')
class Child(Parent):
  def parentMethod(self):
    print ('Calling overriden parent method')
  def childMethod(self):
    print ('Calling child method’)

c = Child()
c.childMethod()
c.parentMethod()
p = Parent()
p.parentMethod()
p.childMethod()
```

Output:

```
Calling parent constructor
Calling child method
Calling overriden parent method
Calling parent constructor
Calling parent method
Traceback (most recent call last):
  File "test.py", line 30, in <module>
    p.childMethod()
AttributeError: 'Parent' object has no
attribute 'childMethod'
```

https://www.tutorialspoint.com/python/python_classes_objects.htm