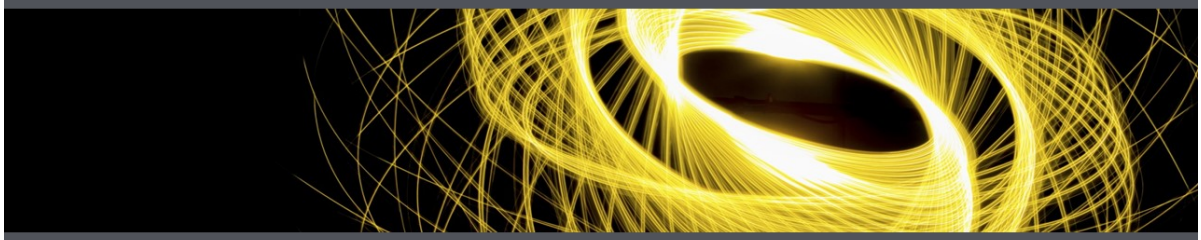
Department of Computer Science

University of Reading

CSMAD21 – Applied Data Science with Python



Functional and Object-Oriented Programming

University of Reading

Lecture Objectives

- Understand and implement the concept of functional programming in Python.
- Define and call functions.
- Comprehend the definition of Mathematical, Pure Functions, Anonymous Functions and Higher Order Functions.
- Understand and implement the concept of Object Oriented Programming in Python.

University of Reading

Outline

- Functional Programming
  - Defining Functions
  - Calling Functions
  - Assignment and Scope
  - Mathematical Function
  - Pure Functions
  - Function as Values
  - Anonymous Functions
  - Higher Order Functions
  - List Comprehensions
  - Closures
  - Generators
- Object Oriented Programming
  - Classes
  - Class Variables
  - Operators
  - Encapsulation
  - Duck Typing

University of Reading

Defining Functions

- A function is a set of statements that take certain inputs, perform especific tasks and produces an output. The general objective of a function is to use this blocks of code instead of rewrite code over again.

- Functions are defined in Python using ***def*** _ followed by the ***function name*** and, if needed, the function arguments between **()**.

In [1]:
```
def hello():
    return 'hello world'
```

In [2]:
```
hello()
```

Out[2]:
```
'hello world'
```

In [3]:
```
hw = hello()
#hw
```

In [4]:
```
hw
```

Out[4]:
```
'hello world'
```

University of Reading

Defining Functions

- Here x is the function argument, and square is the name of the function.
- Several things are different than C and Java:
  - No need to define a return type
  - Parameters have no types defined
  - Various ways of passing parameters

In [5]:
```
def square(x):
    return x*x
```

In [6]:
```
square(4)
```

Out[6]: 16

In [7]:
```
a = square(5)
a
```

Out[7]: 25

University of Reading

Calling Functions

- Arguments can be passed to functions using keywords.
- This allows arguments to be (optionally) explicitly identified.
- We can also specify a default value for an argument, that allow it to be omitted
- This is often seen in data science libraries, for e.g. specifying sensible defaults for algorithm settings:

- `sklearn.decomposition.KernelPCA(n_components=None, kernel='linear', gamma=None, degree=3, coef0=1, kernel_params=None, alpha=1.0, fit_inverse_transform=False, eigen_solver='auto', tol=0, max_iter=None, remove_zero_eig=False, random_state=None, copy_X=True, n_jobs=1`

In [8]:
```
def sqnorm(x,y):
    return x*x+y*y
```

In [9]:
```
a = sqnorm(y=3,x=4)
b = sqnorm(4,y=3)
c = sqnorm(4,3)
```

In [10]:
```
a, b, c
```

Out[10]: (25, 25, 25)

In [11]:
```
#specify a default value for an argument, that allow it to be omitted:

def sqnorm(x,y=0):
    return x*x+y*y
```

In [12]:
```
a = sqnorm(4)
b = sqnorm(4,2)
```

In [13]:
```
a, b
```

Out[13]: (16, 20)

University of Reading

Calling Functions

- Arguments are passed to functions depending on their type:
  - Immutable values such as integers and strings are passed by **value**.
  - Mutable values such as lists are passed by **reference**

- Integers are passed by value:

In [14]:
```
def square(x):
    x = x*x
    return x
```

In [15]:
```
s = 3
square(s)
print(s)
```

```
3
```

- Lists are passed by reference:

In [16]:
```
def listswap(a):
    tmp = a[1]
    a[1] = a[0]
    a[0] = tmp
```

In [17]:
```
l = [1,2]
listswap(l)
print(l)
```

```
[2, 1]
```

University of Reading

Assignments and Scope

- Variables in Python need to be assigned before they can be used. The assignment of a variable determines its scope, i.e. where it can be accessed from.
- Assignments made at the top level are global and accessible from anywhere.
- Assignments made in for or while loops are also in scope at the level the loop is defined at.
- Variables assigned within functions are local to that function definition:
- References to objects will first search the local scope, then global.
- Note that if a variable is assigned in the local scope, the local version will be found:

In [18]:
```
a = 2
def add_a(x): # a is in scope here
    return x+a
```

In [19]:
```
add_a(3)
```

Out[19]:
```
5
```

In [20]:
```
a = 5
def f():
    a = 10 # Local definition of a only in scope inside f
    a = a + 2
    return a
```

In [21]:
```
print(f())
print(a)
```

```
12
5
```

In [22]:
```
#- References to objects will first search the local scope, then global:
a=[]
def f():
    a.append(10) # Finds a in global scope
```

In [23]:
```
f()
print(a)
```

```
[10]
```

In [24]:
```
# - Note that if a variable is assigned in the local scope, the local version will b
a=[]
def f():
    a = []
    a.append(10) #Modifies local a
    return a
```

In [25]:
```
print(f())
print(a)
```

```
[10]
[]
```

University of Reading

Mathematical functions

- Definition:
  - f(x) has one value
  - For a given x, f(x) is always the same
  - Multiple x can give the same value of f(x)
  - Note that functions in programming languages **often do not behave as mathematical functions**.

University of Reading

Mathematical functions

- Which of these functions behaves like a mathematical function? (For a given value of the random seed, the sequence of "random" numbers will always be the same)

```
x = [0]
def incr(y):
        x[0] = x[0] + 1
        return x[0] + y
def unif(a,b):
        # returns a random number
        t = random.random()
        return (b-a)*t+
```

```
def incr(x,y):
        return x + 1 + y
def unif(a,b,seed):
        # sets the random seed
        random.seed(seed)
        # returns a random number
        t = random.random()
        return (b-a)*t+a
```

In [26]:
```
x = [0]
def incr1(y):
    x[0] = x[0] + 1
    return x[0] + y
```

In [27]:
```
def incr2(x,y):
    return x + 1 + y
```

Which of these functions behaves like a mathematical function?

In [28]:
```
print(incr1(1))
print(incr2(0,1))
```

```
2
2
```

In [29]:
```
import random
def unif1(a,b): # returns a random number
    t = random.random()
    return (b-a)*t+a
```

In [30]:
```
unif1(5,6)
```

Out[30]: 5.927594054331711

In [31]:
```
import random
def unif2(a,b): # sets the random seed
    random.seed(6) # returns a random number
    t = random.random()
    return (b-a)*t+a
```

In [32]:
```
unif2(5,6)
```

Out[32]: 5.793340083761663

Which of these functions behaves like a mathematical function?

(For a given value of the random seed, the sequence of "random" numbers will always be the same)

In [33]:
```
print(unif1(5,6))
print(unif2(5,6))
```

```
5.821954042319727
5.793340083761663
```

University of Reading

Pure functions

- In programming, pure functions:
  - Always produce the same output when given the same arguments.
  - Have no side effects (no modification of any variables outside of those local to the function, or I/O).
- This restricts functions to behave like mathematical functions.
- In Python there is no way to enforce that a function behaves in this way, but when using functions as arguments to higher order functions it is a good idea to stick to pure functions.

University of Reading

Functions as values

- In many programming languages, functions can be treated as values, and passed to other functions.
- This means you can do many interesting things:
  - Write functions that apply arbitrary operations based on their parameters.
  - Write functions that generate a function and return it as a value.

In [34]:
```
def g(x):
    return (2*x)
```

```
def map1(f,xs):
    result = []
    for x in xs:
        t = f(x)
        result.append(t)
    return result
```

In [35]:
```
l = [3,6,9,12]
map1(g,l)
```

Out[35]: [6, 12, 18, 24]

University of Reading

Anonymous functions

- To quickly generate functions in Python, anonymous functions can be built using the **lambda** keyword.
- Although no name is assigned to the function when it is defined, you can assign the function to a variable just like you would with any other value.

University of Reading

Aside - lambda calculus

- Why is it called a lambda?
- In many programming languages anonymous functions are referred to as lambdas. This is because of something called the lambda calculus. The lambda calculus can compute anything that a Turing machine can.
- In the lambda calculus we could write this function:

```
lambda x: x*x+1
```

- In the lambda calculus as: λx.x2 +1 (λ is the Greek letter lambda)

University of Reading

Aside - lambda calculus

- Functions application in the lambda calculus is written as:

(λx.x2 +1)5

- Which is equivalent to:

```
(lambda x: x*x+1)(5)
```

- The key operation in lambda calculus is substituting some variable, here x, with a value, here 5, in an expression.

In [36]:
```
lambda x: x*x+1
(lambda x: x*x+1)(5)
```

Out[36]: 26

In [37]:
```
## Lets redo our Function g as a lambda function:
## We can send the lammda function as a parameter tu value function.
l = [3,6,9,12]
map1(lambda x: 2*x,l)
```

Out[37]: [6, 12, 18, 24]

In [38]:
```
#Although no name is assigned to the function when it is defined, you can assign the
f = lambda x: x if x>0 else -x
f(-6)

#A shorthand for if statements is __A if condition else B__.
```

Out[38]: 6

University of Reading

Higher order functions

- Higher order functions are functions that take other functions as arguments and operate on them, or that return functions as results.
- In programming, higher order functions can be used to write customisable functions, whose operations depend on their arguments. They can also be used to build functions.

University of Reading

Higher order functions - Maps

- The map operation applies a function to every element of a collection.
- Generally we want to use pure functions, as they have no side effects, i.e. functions that behave as mathematical functions. For example generally in a map, the order in which the elements will be processed is not defined, and may even be performed in parallel on multiple CPU cores or different computers.
- Why maps?
  - Readability – Makes it clear you are applying the same function to each list member
  - Correctness – Combine existing built-in functions
  - Parallelisable – Each function can be applied in a different CPU thread, or on a different computer (see Map Reduce).

In [39]:

```
def map1(f,xs):
    result = []
    for x in xs:
        t = f(x)
        result.append(t)
    return result
```

An example – finding the average (mean) of each element of a list of lists:

In [40]:

```
def mean(xs):
    n = len(xs)
    if n>0:
        s = sum(xs)
        return s/n
    else:
        return 0.0
```

In [41]:
```
map1(mean,[[1,2,3],[5,5,4,5],[9,8,3,4,5],[]])
```

Out[41]:
```
[2.0, 4.75, 5.8, 0.0]
```

University of Reading

Higher order functions - Filter

- The filter operation applies a condition to each member of a collection and returns a list only containing members for which the condition was true.
- The function passed to filter should return True or False.
- Why maps?
  - Readability – Makes it clear you are applying the same function to each list member
  - Correctness – Combine existing built-in functions
  - Parallelisable – Each function can be applied in a different CPU thread, or on a different computer (see Map Reduce).

In [42]:
```
def filter1(p,xs):
    result = []
    for x in xs:
        if p(x):
            result.append(x)
    return result
```

Filtering values from a list:

In [43]:
```
l = [-3,4,2,-1,0,-9,5]
filter1(lambda x: x>0,l)
```

Out[43]:
```
[4, 2, 5]
```

We can combine maps and filters:

In [ ]:
```

```

In [44]:
```
l = [-3,4,2,-1,0,-9,5]
filter1(lambda y: (y % 2 == 0), map1(lambda x: x*x,l))
```

Out[44]:
```
[16, 4, 0]
```

In [ ]:
```

```

Functional programming with the Python Standard Library

University of Reading

- **Map**
  - Python has a built-in function map that behaves similarly to our map function defined above. The difference is that it can take any number of iterable objects as inputs
- **Operators as functions**
  - For various built-in operators in Python the operator module contains function definitions that can be passed to functions like map
- **Filter**
  - There is also a built-in filter function in the standard library.

In [45]:
```
##First we need to define a function
def sub(x,y):
    return x-y
```

In [46]:
```
a = map(sub,[1,2,3],[3,2,1])
list(a)
```

Out[46]:
```
[-2, 0, 2]
```

Notice that here a is not actually a list but a generator, an iterable object. We use list(a) to evaluate a and convert it to a list.

In [47]:
```
type(a)
```

Out[47]:
```
map
```

Operators as functions

Maps

For various built-in operators in Python the operator module contains function definitions that can be passed to functions like map. The difference is that it can take any number of iterable objects as inputs:

In [48]:
```
import operator
a = map(operator.sub,[1,2,3],[3,2,1])
list(a)
```

Out[48]:
```
[-2, 0, 2]
```

In [49]:
```
type(a)
```

Out[49]:
```
map
```

Filter

There is also a built-in filter function in the standard library:

In [50]:
```
a = filter(lambda x:x%3==0,range(10))
list(a)
```

Out[50]:
```
[0, 3, 6, 9]
```

University of Reading

List comprehensions

- A quick way to apply some operations to some group of objects is to use a list comprehension. This allows you to build a list by specifying how it should be constructed.

In [51]:
```
a = [ x * x for x in range(1,11)]
a
```

Out[51]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

You can also apply filtering to the for loop:

In [52]:
```
b = [ x * x for x in range(1,11) if (x*x)%3==1 ]
b
```

Out[52]:
```
[1, 4, 16, 25, 49, 64, 100]
```

You can unpack tuples in list comprehensions

In [53]:
```
a = [ (4.0,1.2),(5.6,7.1),(1.1,0.6)]
b = [(x-y)*(x-y) for x,y in a]
b
```

Out[53]:
```
[7.839999999999999, 2.25, 0.2500000000000001]
```

You can also nest for loops inside of the list comprehension:

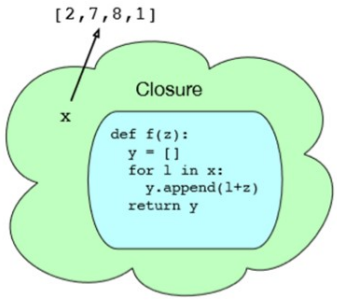
In [54]:
```
c = [(x,y) for x in range(1,3) for y in range(1,4)]
c
```

Out[54]: [(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3)]

University of Reading

Closures

- Functions can also be grouped together with the environment they were defined in. This is known as a closure.
- This allows you to do things like define functions that build and return other functions.
- Global or local variables of any type can be captured in a closure.
- However if the variable captured is a list or dictionary, the closure will store a reference to it, not a copy.



21

In [55]:
```
def addXMaker(x):
    def f(y):
        return x + y
    return f
```

In [ ]:
```

```

In [56]:
```
g = addXMaker(3)
g(5)
```

Out[56]: 8

Global or local variables of any type can be captured in a closure:

In [57]:
```
def g():
    x = [2,7,8,1]
    def f(z):
        y = []
        for l in x:
            y.append(l+z)
        return y
    return f
```

In [58]:
```
closure = g()
closure(4)
```

Out[58]: [6, 11, 12, 5]

However if the variable captured is a list or dictionary, the closure will store a reference to it, not a copy.

In [59]:
```
def g():
    x = [2,7,8,1]
    def f(z):
        y = []
        for l in x:
            y.append(l+z)
        return y
    x[2] = 9
    print(x)
    return f
```

In [60]:
```
closure = g()
closure(4)
```

```
[2, 7, 9, 1]
```

Out[60]:
```
[6, 11, 13, 5]
```

University of Reading

Generators

- Generators can be used to lazily construct iterable sequences of objects. A simple way of constructing a generator in Python is to use a syntax very similar to a list comprehension.

```
a = (pow(x,5) for x in range(pow(10,128)))
```

- If we actually evaluated this as a list comprehension it would run out of memory constructing the list before completing. A generator instead produces an object that can be used to iterate over the list, **only on demand.**

In [61]:
```
#a = [pow(x,5) for x in range(pow(10,128))]
#a
```

In [62]:
```
import itertools
a = (pow(x,5) for x in itertools.count(1))
next(a)
```

Out[62]:
```
1
```

In [63]:
```
next(a)
```

Out[63]:
```
32
```

In [64]:
```
next(a)
```

Out[64]: 243

In [65]:
```
next(a)
```

Out[65]: 1024

next(a) takes a value from the generator, and updates it to step to the next value in the sequence.

count is a function from the itertools module that returns an iterator that counts up from a starting value

---

University of Reading

Why?

- These concepts will appear a lot in data science applications, e.g.:
  - Map a function over columns of a data set (e.g. calculate mean of each column)
  - Filter a list of values that match some condition
- The two most common data science languages, Python and R, both support functional programming.
- Recognising common patterns that can be implemented using higher order functions helps you write more concise, readable and reliable code.

University of Reading

Object-oriented programming

- Encapsulation
  - Data and the functions that work on them are bound together in objects. We might want to hide variables from view outside of the object.
- Composition
  - Objects provide abstraction and can be used as building blocks to construct more complex objects.
- Inheritance
  - Objects can be derived from other objects in a hierarchy.

University of Reading

Classes

- Class definitions are simple, methods are defined as functions inside of the class definition.
- Classes are just collections of Python functions and assignments. When an object of a class is instantiated a new object of that class is created.
- To allow our methods to work with distinct **instances** of objects, methods take an extra first argument, by convention called **self**. This is a reference to the instance of the object that the method was called on.

In [66]:
```
#First Class
class MyClass1:
    # An instance method
    def set(self,value):
        # A variable defined in each instance
        self.val = value
    def get(self):
        return self.val
```

- This class has a variable val that belongs to each instance of the class, and can be modified using the set and get methods. We can create an instance of the class using:

In [67]:
```
a = MyClass1()
a
```

Out[67]:
```
<__main__.MyClass1 at 0x25991665198>
```

So if we call set on a MyClass object:

In [68]:
```
a = MyClass1()
a.set(5) # Calls MyClass.set(a,2)

a.get()
#The actual function call is to MyClass.set(a,...).
```

Out[68]:
```
5
```

University of Reading

Instance initialisation

- Class instances in Python are initialised using a special function called __init__. This is called when the constructor of the class is invoked.
- If you leave it blank then Python will just create an object of that class with no initialisation.

- **Class variables**
  - Variables can be defined inside of a class, these assignments will be made when the class is defined, and are shared attributes between all instances of the class.

In [69]:
```
class MyClass2:
    def __init__(self,value):
        self.val = value
        # A class method
    def set(self,value):
        # A variable defined in each instance
        self.val = value
    def get(self):
        return self.val
```

In [70]:
```
MyClass2(6)
```

Out[70]:
```
<__main__.MyClass2 at 0x25991665710>
```

In [71]:
```
#Using the default constructor, or specifying arguments to the __init__ function:
storeA = MyClass1()
storeA.set(32)
#print(storeA.get()) # 32

storeB = MyClass2(64)
print(storeB.get()) # 64
print(storeA.get()) # 32
```

```
64
32
```

In [1]:
```
##Class Variables

class MyClassCounter:
    readCounter = 0 # A class variable
    writeCounter = 0 # A class variable
    def set(self,value): # A class variable for *all* instances.
        MyClassCounter.writeCounter += 1
        self.val = value
    def get(self): # A class variable for *all* instances.
        MyClassCounter.readCounter += 1
        return self.val
```

In [2]:

```
storeA = MyClassCounter()
storeA.set(32)
print(storeA.get()) # 32
print(MyClassCounter.readCounter) # 1
print(MyClassCounter.writeCounter) # 2
```

```
32
1
1
```

In [3]:

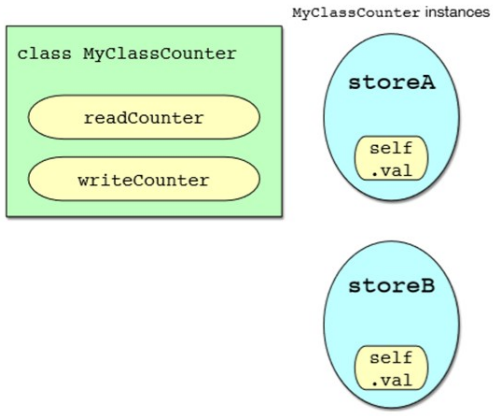
```
storeB = MyClassCounter()
storeB.set(16)

print(MyClassCounter.readCounter) # 1
print(MyClassCounter.writeCounter) # 2
```

```
1
2
```

- Notice that the readCounter and writeCounter variables are shared by both storeA and storeB.

University of Reading

Classes examples



University of Reading

Operators

- As we saw with __init__ Python uses special methods in classes for certain language functionality. As well as constructors, you can specify how objects of your class should behave when used with various Python language operators.

Method	Python syntax	Description
__init__(self,args)	MyClass(args)	Object instantiation
__add__(self,b)	a + b	Addition
__sub__(self,b)	a - b	Subtraction
__contains__(self,b)	b in a	Membership

- Operators like + and - don't necessarily have to perform mathematical addition or subtraction in some way – you can use them as syntactic shortcuts for different operations (e.g. concatenation).

In [75]:
```
class MyClassAdd:
    def __init__(self,value):
        self.val = value
    def set(self,value):
        self.val = value
    def get(self):
        return self.val
    def __add__(self,x):
        return MyClassAdd(self.val+x.get())
```

In [76]:
```
a = MyClassAdd(2)
b = MyClassAdd(3)

a + b
c = a + b
c
c.get()
```

Out[76]: 5

If "a + b" and a is an instance of MyClassAddd, then Python will look the class definition of MyClassAddd. If MyClassAddd has a method **add** it will be called with x.**add**(y)

University of Reading

Encapsulation

- When a class is defined, all attributes and methods are **public by default**, meaning that any attributes of a class instance can be modified outside of the class.
- There are advanced approaches to get around this – for now simply be careful when working with classes!

In [77]:
```
class MyClass2:
    def __init__(self,value):
        self.val = value
        # A class method
    def set(self,value):
        # A variable defined in each instance
        self.val = value
    def get(self):
        return self.val
```

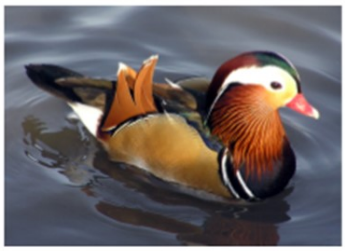
In [78]:
```
a = MyClass2(3)
a
a.val = 4
print(a.get())
```

```
4
```

University of Reading

Duck typing

- In Python, we are able to use instances of classes without specifying that a variable must be an object of that specific class. In fact if we write some code that uses objects with a .get() method, any Python object with a .get() method can be used, regardless of which class it belongs to.
- This means that we can define classes whose objects behave exactly like objects of another class, without inheritance. This is known as duck typing.



In [79]:
```
#This function will work with instances of any class that have get and set methods:

def addToStore(s,x):
    value = s.get()
    s.set(value+x)
```

In [80]:
```
#So we can use it with both MyClass and MyClassCounter:

a = MyClass2(4)
b = MyClassCounter()
b.set(3)

addToStore(a,3)
addToStore(b,3)
```

In [81]:
```
a.get(), b.get()
```

Out[81]: (7, 6)

University of Reading

Summary

- Functions can be treated as values.
- Higher order functions allow common programming patterns to be used and composed easily.
- Supported by the Python standard library.
- Python supports different programming paradigms such as functional and object oriented.

University of Reading

Questions

University of Reading

References

- Thomas McKinney, Wes ‘Python for data analysis : data wrangling with Pandas, NumPy, and IPython’.

In [ ]:

In [ ]: