Advanced School in Artificial Intelligence

Introduction to Python

Ing. Zese Riccardo
riccardo.zese@unife.it

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Regione Emilia-Romagna

Università degli Studi di Ferrara

Outline

- Introduction to Python
- Introduction to Neural Networks
- Convolutional NN
- Recurrent NN
- Autoencoders and self supervised learning

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Outline

- Introduction to Python
- Introduction to Neural Networks
- Convolutional NN
- Recurrent NN
- Autoencoders and self supervised learning

Functions

- The definition of a function is

```
def  f(x=10):
     print(x)


f()
f(5)
```

- Arguments are 0 or more variable. Variables can have a default value assigned.
- The program will print 10 and 5

Functions

- The name of a function:
  - Must start with a letter or a an underscore
  - Can include only, letters, underscores and numbers

  - As for variables, naming convention requires to use only lower case letters and divide the words by underscore.

Functions

- We can call functions calling other functions

```
sum(divide(a,b),multiply(c,d))
```

- It is used an inside-out direction.
- Same when considering local and global variables:
  - Local variables are defined inside of functions, global variables are defined outside of functions.
  - First, check local variables defined in a function.
  - Then it will check global variables.
  - Finally it will check built-in variables.

More on function definition

- We have already seen that in the definition of a function it is possible to define some default argument values.

```
def useless_f(x, y, n=5, s='boh'):
    print(x, y, n, s)
```

- This function can be called in many ways:
  - Giving only mandatory arguments `useless_f(1,2)`
  - Giving one optional argument `useless_f(1,2,3)`
  - Giving all arguments `useless_f(1,2,3,4)`
  - … and more

Must follow the order

Default arguments

- We have to pay attention about default arguments

```
i = 5

def f(arg=i):
    print(arg)

i = 6
f()
```

The output will be 5

Default arguments

- We have to pay attention about default **mutable** arguments

```
def f(a, L=[]):
    L.append(a)
    return L
```

`print(f(1))`	The output will be [1]
`print(f(2))`	The output will be [1,2]
`print(f(3))`	The output will be [1,2,3]

Default arguments

- We have to pay attention about default **mutable** arguments.
- If we want to maintain each call separated we must modify the function

```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

`is` for now is equivalent to `==`

More on function definition

```
def useless_f(x, y, n=5, s='boh'):
    print(x, y, n, s)
```

- This function can be called in many ways:
  - Giving only mandatory arguments `useless_f(1,2)`
  - Giving one optional argument `useless_f(1,2,3)`
  - Giving all arguments `useless_f(1,2,3,4)`
  - Giving pairs key-value `useless_f(s=3,x=1,y=2,n=4)`

Using this way we do not have to follow the order of arguments.
We can give all or only some optional arguments.
ATTENTION: `useless_f(1,2,n=4)` is valid while
`useless_f(1,2,s=3,4)` is **not**

Functions vs methods

- Recall that methods are similar to functions but are linked to the type of object.
- They can be called on the object via the dot notation.

```
>>> s = "Python"
>>> s.upper() # this methods returns "PYTHON"
>>> s.lower() # this method returns "python"
```

Functions vs methods

- Functions belong to modules.
- Methods belong to objects.

- **`len(str)`** is a function
- **`str.lower()`** is a method.

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Objects

- We have seen that there are objects that are immutable (int, str, etc.)
- Like strings in Java, if we try to modify them we simply create new instances of the object and throw the old ones.
  - This behavior can work fine with simple objects but not with complex, large ones
  - We need **mutable objects**

Mutable Objects

- If we want to change a really large object without keeping the original, then making a big copy, modifying it and throwing the original is wasteful.
- Instead, we can use a mutable object, that we're allowed to change.
- This also allows us to define functions that change objects, rather than returning new ones.

Classes

- Besides modules and functions, Python allows also the definition of classes, following the object-oriented programming paradigm.
- In Pyhton a class is a new type, using a class we can instantiate objects of that class.

Python Classes in Brief

```
class MyClass (Super, Classes):
   def method_name(self,arguments):
      # statements
   # other definitions
```

Method definition

Optional. Every class inherits from `Object`.
Support to **multiclass inheritance**.
The search of methods and attributes is performed following a certain order and avoiding searching in same classes more time, called **Method Resolution Order (MRO)**.

To create an instance: `x = MyClass()`

To access an attribute: dot notation → `x.attribute_name`

To call a method: dot notation → `x.method_name(arguments)`

Python Classes in Brief

```
class Point:
    '''
    This is a 2-D point. Attributes: x,y
    '''
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
    def distance_from_origin(self):
        return ((self.x**2) + (self.y**2))**(1/2)

p = Point(1)
dist = p.dinstance_from_origin()
print(dist)
```

Class definition

Class description (optional)

Constructor method

Other methods. **NOTE: every** method must have **`self`** as first argument.

Use of default value fo **`y`**

No need to pass self reference **`p`**, but only other arguments (if there are any)

Classes

- A class is defined using the **`class`** keyword in the following way:

```
class MyClass:
    # definitions
```

Classes

- Forget for a while inheritance, once we define a class

```
class MyClass:
    # definitions
```

- we can simply create an instance of this class in the following way:
  `x = MyClass()`

Classes

- The call of `x = MyClass()` creates a new emtpy object.
- If we want to initialize attributes of the class we need to specify a constructor.
- We can use the method `__init__()` that is automatically invoked during the creation of the object.
- We can specialize the method `__init__()` passing it some arguments.
    - ATTENTION: only one constructor can be defined. Overloading in Python is handled by the use of arguments having default values.

Regione Emilia-Romagna

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Università degli Studi di Ferrara

Classes: an example

```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
```

- This class defines a 2D point by means of two attributes, namely **x** and **y**.
- In the definition of methods, we have to pass **self** as first argument ensuring that the operations described in the method are performed on the specific instance.

Classes: an example

```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
```

- The use of arguments with default values “simulates” the implementation of more constructors

```
p1 = Point()       # passing no values
p2 = Point(1)      # passing x
p3 = Point(1, 2)   # passing x,y
p4 = Point(y = 2)  # passing y
```


Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Classes: an example

```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
```

- The use of arguments ” the implementation of more constructors

Note that the use of self as first attribute in the definiton allows us to not pass the instance to the method.

```
p1 = Point()
p2 = Point(1)       # passing x
p3 = Point(1, 2)    # passing x,y
p4 = Point(y = 2)   # passing y
```

Classes

- We have already seen methods working on instances.
- The definition of a method is similar to the definition of a function. One of the differences is that it must be defined inside the class definition.
- Moreover, we know that methods work on the values defining the state of the instance → classes have attributes defining their state.
- An attribute is accessible via the dot notation → `p1.x`

Methods

- The definition of a method is

```
def method_name(self,arguments):
      # statements
```

- Must be done inside a class
- Arguments are 0 or more variables.
- The body of the function (statements) must be indented.
- If the block contains the keyword **return**, it returns a value; otherwise it returns the special value **None**.
- To refer to object’s attributes use `self`.

Classes: an example

```
class Point:
    '''
    This is a 2-D point
    Attributes: x,y
    '''

    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def distance_from_origin(self):
        return ((self.x**2) +
                (self.y**2))**(1/2)
```

Inheritance

- As normal in a OOP language, Python supports inheritance between classes.
- The superclass must be given in the definition of the class.

```
class MyClass(ItsSuperclass):
    # definitions
```

Inheritance

- Note that also classes that do not define a superclass in their definition, in reality inherit from a class.

```
class MyClass(object):
    # definitions
```

- Every class is a subclass of the type **`object`**
- If we omit the superclass, this will be **`object`**

Inheritance

- As superclass we can use arbitrary expressions. This can be useful, for example, when the base class is defined in another module

```
class MyClass(mymod.MySuperClass):
    # definitions
```

- There are no special mechanisms for constructors.
- Search for methods/attributes is performed desceding the chain of superclasses

Multiclass inheritance

- Python supports multiple inheritance. A class definition becomes

```
class MyClass(Class1,Class2):
    # definitions
```

- Search for attributes inherited from a parent class as almost depth-first, left-to-right, not searching twice in the same class where there is an overlap in the hierarchy.
  - If an attribute is not in **`My_class`** it is searched in **`Class1`** (if not present there, in its superclasses). If not present in **`Class1`** and superclasses, it is searched in **`Class2`** and superclasses.

Multiclass inheritance

- The search of methods and attributes is performed following a certain order and avoiding searching in same classes more time, called **Method Resolution Order (MRO)**.
- You can view the MRO by using `__mro__` attribute.
- The first correspondence found stops the search.

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Multiclass inheritance

```
class A:
    pass

class B(A):
    pass

class C(A):
    pass

class E(B,C):
    pass

print(E.__mro__)
```

Output:

```
(<class '__main__.E'>,
 <class '__main__.B'>,
 <class '__main__.C'>,
 <class '__main__.A'>,
 <class 'object'>)
```

Multiclass inheritance

```
class A:
    pass

class B(A):
    pass

class C(A):
    pass

class E(B,C):
    pass

print(E.__mro__)
```

Output:

```
(<class '__main__.E'>,
 <class '__main__.B'>,
 <class '__main__.C'>,
 <class '__main__.A'>,
  <class 'object'>)
```

As one can see, Python does not always follow a depth-first, left-to-right strategy. It first creates the complete chain

E – B – A – Object – C – A – Object

then removes classes (left-to-right) such that there is other class in the tail of the search path which inherits from it. In this case it is more natural to use the method defined by its derived class.