Objects and Classes

```
          #       In Python, everything is an object.
          #       objects are created form classes (we come to the "create from" process soon).
          #       Every object is of a certain type.
          #       The type of an object tells, from which class the object was created.
          #       Some examples:
OC001 >>>     objects = 7, 3.5, 'abc', True, 7==3, None, max, [1,3], (2,3), float
OC002 >>>     for item in objects:
        ...           print("{} is of type: {}".format(str(item), type(item)))
        p()       7 is of type: <class 'int'>
        p()       3.5 is of type: <class 'float'>
        p()       abc is of type: <class 'str'>
        p()       True is of type: <class 'bool'>
        p()       False is of type: <class 'bool'>
        p()       None is of type: <class 'NoneType'>
        p()       <built-in function max> is of type: <class 'builtin_function_or_method'>
        p()       [1, 3] is of type: <class 'list'>
        p()       (2, 3) is of type: <class 'tuple'>
        p()       <class 'float'> is of type: <class 'type'>
```

Definition of a class

```
        #     A class is like a cookiecutter or a stamp. The cookie and the print on the paper
        #     are objects, created from their 'class'

        #     Let's define a class:
OC003 >>>     class Book():
      ...         def what(self):
      ...             print('a Book')

        #     The Book class has just one simple method.
        #     Let's create book objects:
OC004 >>>     paperback = Book()
OC005 >>>     dictionary = Book()
OC006 >>>     schoolbook = Book()

OC007 >>>     type(paperback), type(dictionary)
      ==>     (<class '__main__.Book'>, <class '__main__.Book'>)
        #     all books have the what() method
OC008 >>>     paperback.what()
      p()     a Book
OC009 >>>     schoolbook.what()
      p()     a Book
```

CLasses need initialization

```
        #    A class is considered a 'data type' with some defined 'behavior'
        #    To start with the data part: There must be an initialization method
OC010 >>>    class Book():
      ...        def __init__(self, isbn, author, title):
      ...            self.isbn = isbn
      ...            self.author = author
      ...            self.title = title
      ...        def get_isbn(self):
      ...            return self.isbn

        #    first an example, how this is used
OC011 >>>    mybook = Book('1234-5678-90','Summerfield, Mark', 'Programming in Python')
OC012 >>>    mybook.get_isbn()
      ==>    '1234-5678-90'

        #    To write 'Book(p1, p2, p3)' creates a new object. It is like calling a function.
        #    'mybook = Book(p1, p2, p3)' assigns the new object to a variable
        #    When a new object is created, the '__init__()' method is called.
        #    Arguments for the class creation are given to the __init__() method
```

Object creation - 'me, myself, I'

```
      #     all methods of an object automatically have added one first argument
      #     the first argument is always called 'self'. 'self' is the object itself.

      #     'self.isbn = isbn' assigns an attribute to the self object
OC013 >>>   show_attr(mybook)       # the show_attr() function is not available outside this presentation
    p()     attr isbn: '1234-5678-90'
    p()     attr author: 'Summerfield, Mark'
    p()     attr title: 'Programming in Python'

      #     The get_isbn() method is also called with the 'self' argument
      #     It uses 'self' to access the 'isbn' attribute
```

Comparing Objects

```
        #    Quick define another class:
OC014 >>>    class Box():
      ...        def __init__(self, content=None):
      ...            self.content = content
      ...        def what(self):
      ...           print("box contains:", self.content)
        #    create two box objects
OC015 >>>    box1 = Box('shoes')
OC016 >>>    box2 = Box('socks')
OC017 >>>    box2 == box1   # compare the two - no surprise
      ==>    False
OC018 >>>    box3 = Box('shoes')
OC019 >>>    box3 == box1   # compare the two - hmm, they seem to be equal
      ==>    False
        #    try this:
OC020 >>>    box3 = box1   # assignment
OC021 >>>    box3 == box1   # comparison
      ==>    True
OC022 >>>    box3 is box1   # check for identity
      ==>    True

        #    we can make objects smarter ...
```

Comparing Objects

```
OC023 >>>   class Box():
      ...       def __init__(self, content=None):
      ...           self.content = content
      ...       def is_equal(self, other):
      ...          return self.content == other.content

        #   create two box objects
      ...   box1 = Box('shoes')
OC024 >>>   box2 = Box('socks')
OC025 >>>   box2.is_equal(box1)   # compare the two with a dedicated method
      ==>   False
OC026 >>>   box3 = Box('shoes')
OC027 >>>   box3.is_equal(box1)   # compare the two - now the comparison works
      ==>   True

        #   but true: the 'box1 == box2' looks much nicer than the 'box1.is_equal(box2)'
        #   we can have it!
```

Objects and Magic Methods

```
OC028 >>>   class Box():
      ...       def __init__(self, content=None):
      ...           self.content = content
      ...       def __eq__(self, other):     # __eq__ is one of the "magic methods"
      ...          return self.content == other.content
      ...       def __repr__(self):    # another magic method
      ...          return("A Bag, contains {}".format(self.content))

OC029 >>>   box1, box2 = Box('shoes'), Box('shoes')
OC030 >>>   box1 is box2   # not identical
      ==>   False
OC031 >>>   box1 == box2   # but equal
      ==>   True
OC032 >>>   box3     # this is from the previous definition of the Box class: the 'normal' output
      ==>   <__main__.Box object at 0x000000000342A630>
OC033 >>>   box1     # this if from the above definition, which replaces the __rep__ method
      ==>   A Bag, contains shoes

        #   There are a number special methods to allow to define the behaviour of objects
        #   To sort a collection of objects, the class must define all of the __eq__, __ne__,
        #   __gt__, __ge__, __lt__ and __le__ methods
```

Objects and Magic methods - Compare again

```
OC034 >>>   class Bag():
      ...       def __init__(self, content=None):
      ...           self.content = content   # Bag uses the same name as Box - completely legal
      ...       def __eq__(self, other):
      ...          return self.content == other.content   # supposed to compare two bags
OC035 >>>   bag1 = Bag('shoes')
OC036 >>>   bag1 == box1    # comparing apples and pears - should not be allowed
      ==>   True

        #   better also check the type:
OC037 >>>   def cmp(self, other):   # define a comparison, which also checks the type of the arguments
      ...       if type(self) is not type(other):
      ...           raise TypeError("Cannot compare {} and {}".format(type(self), type
      ...       return self.content == other.content
OC038 >>>   Box.__eq__ = Bag.__eq__ = cmp    # This is a stunt! - Kids: Don't try this at home!
        #   this replaced the __eq__ method in both classes, which changes the behaviour of the objects
OC039 >>>   box1 == box2
      ==>   True
OC040 >>>   box1 == bag1     # this is why Python is called a dynamic language
      err!  TypeError("Cannot compare <class '__main__.Box'> and <class '__main__.Bag'>
```

Classes and Objects: The account example

```
OC041 >>>   class Account():
      ...       def __init__(self, account_number):
      ...           self.acct_no = account_number
      ...           self.balance = 0.0   # for simplicity reasons, use type float
      ...       def deposit(self, amount):
      ...           self.balance += amount
      ...       def withdraw(self, amount):
      ...           self.balance -= amount
      ...       def get_balance(self):
      ...           return self.balance

OC042 >>>   my_account = Account(account_number='8761233-2')
OC043 >>>   my_account.deposit(200)
OC044 >>>   my_account.deposit(30.50)
OC045 >>>   my_account.withdraw(85.10)
OC046 >>>   my_account.get_balance()
      ==>   145.4

        #   Objects allow us to create 'models' of things in the real world.
        #   The methods of an object should be like "real world interactions"
```

The Account example - add a transaction trail

```
OC047 >>>   class Account():
      ...       def __init__(self, account_number, start_balance=0.0):
      ...           self.acct_no = account_number
      ...           self.balance = start_balance
      ...           self.start_balance = self.balance
      ...           self.tx_trail = []
      ...       def deposit(self, date, amount, text):
      ...           self.balance += amount
      ...           self.tx_trail.append((date, amount, text))
      ...       def withdraw(self, date, amount, text):
      ...           self.balance -= amount
      ...           self.tx_trail.append((date, -amount, text)) # see the minus sign
      ...       def get_balance(self):
      ...           return self.balance
      ...       def print_transaction_trail(self, p_date):
      ...           curr_bal = self.start_balance
      ...           print("{:10s} {:36s} {:8s} {:8.2f}"
      ...                 .format("", "Start Balance", "", self.start_balance))
      ...           for date, amount, text in sorted(self.tx_trail):
      ...               curr_bal += amount
      ...               print("{} {:36s} {:8.2f} {:8.2f}"
      ...                 .format(date, text, amount, curr_bal))
      ...           print("{} {:36s} {:8s} {:8.2f}"
      ...                 .format(p_date, "Final Balance", "", curr_bal))
```

The Account example - experiment

```
        #   create an account and make some transactions
OC048 >>>   myaccount = Account('1234-5678-90', 420.20)
OC049 >>>   myaccount.deposit('2016-01-03', 200, "conta 23455 de 20.12.2015")
OC050 >>>   myaccount.withdraw('2016-01-05', 25.30, "lidl, compras de 4.1.2016")
OC051 >>>   myaccount.withdraw('2016-01-20', 50.00, "Caixa MB Arroios")
OC052 >>>   myaccount.withdraw('2016-01-12', 12.10, "Pingo Doce,compras de 12.1.2016")
OC053 >>>   myaccount.deposit('2016-01-25', 40.00, "retorno do credito, Michael Müller'

OC054 >>>   myaccount.print_transaction_trail('2016-01-31')
      p()              Start Balance                                   420.20
      p()   2016-01-03 conta 23455 de 20.12.2015              200.00   620.20
      p()   2016-01-05 lidl, compras de 4.1.2016              -25.30   594.90
      p()   2016-01-12 Pingo Doce,compras de 12.1.2016        -12.10   582.80
      p()   2016-01-20 Caixa MB Arroios                       -50.00   532.80
      p()   2016-01-25 retorno do credito, Michael Müller      40.00   572.80
      p()   2016-01-31 Final Balance                                   572.80
```

Wrestling with classes and objects

# The following examples