An Introduction to Python

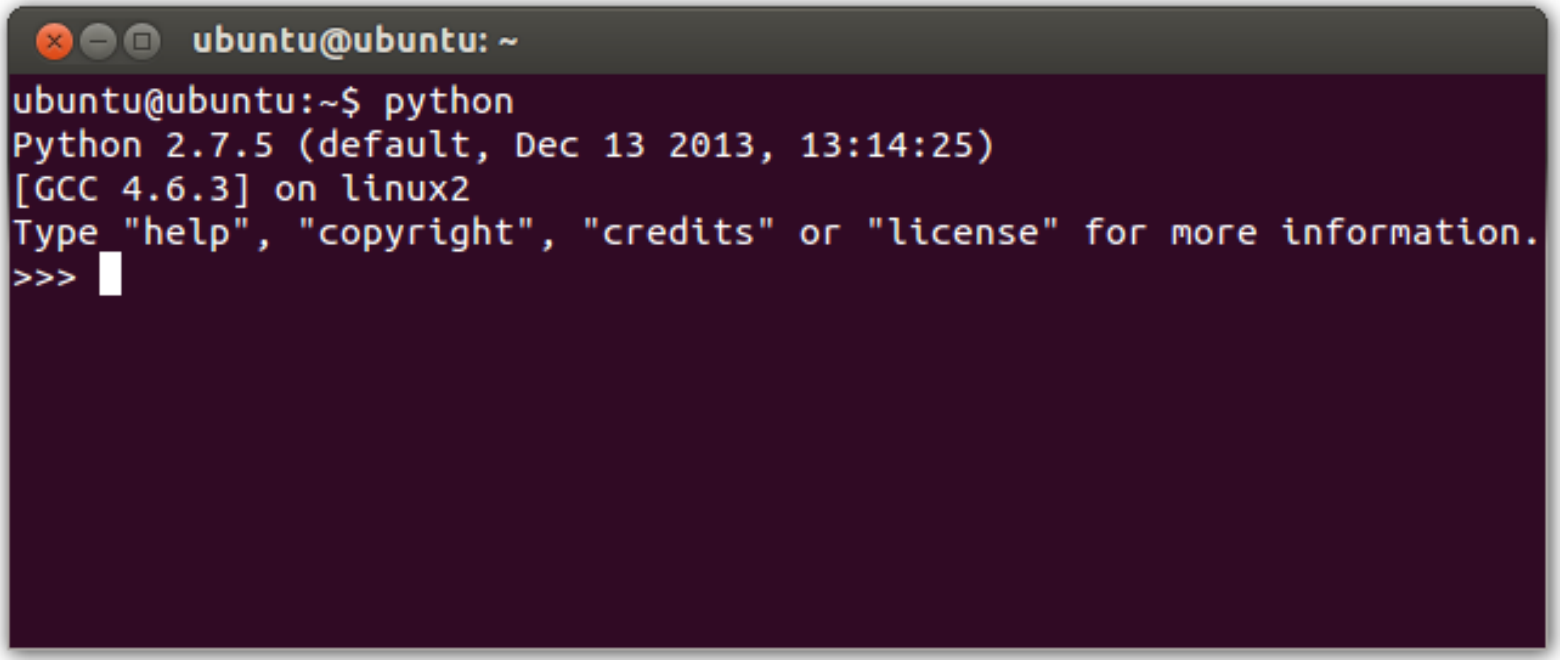
Parham Alvani
Amirkabir University of Technology

parham.alvani@gmail.com
May 14, 2015

What is python...

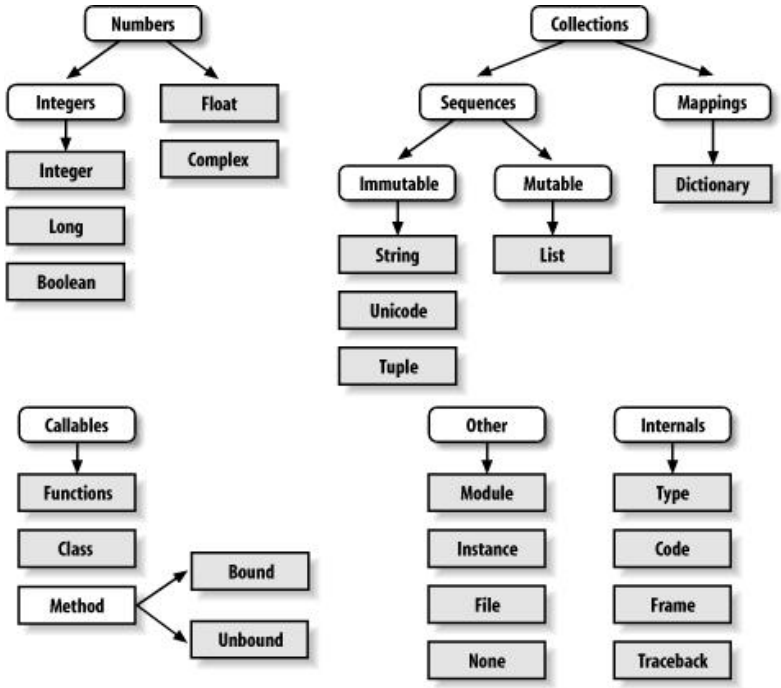
Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable..

Environment



Variable Types

- Numbers
- String
- List
- Tuple
- Dictionary



Numbers

- Number data types store numeric values.
- They are immutable data types, means that changing the value of a number data type results in a newly allocated object.

Strings

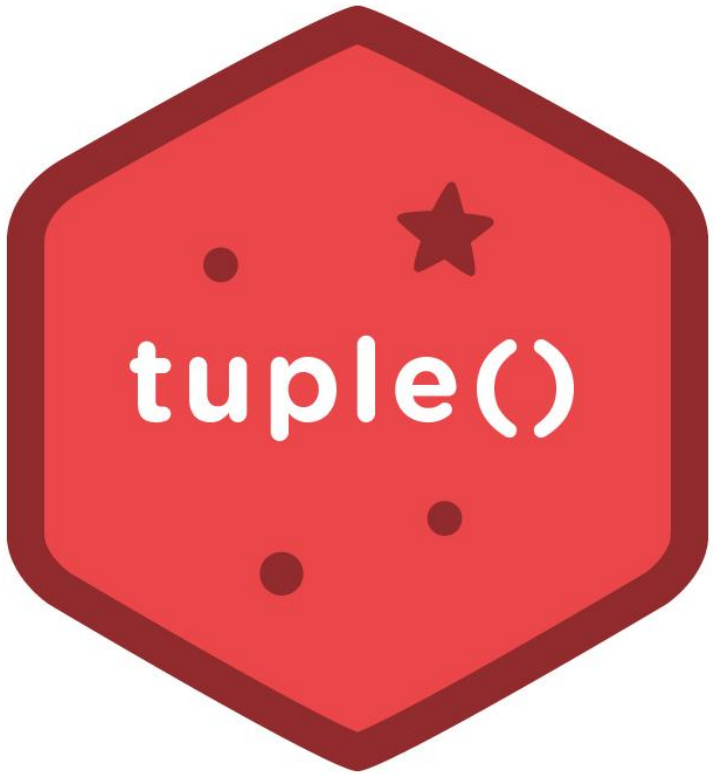
- Strings are among the most popular types in Python.
- We can create them simply by enclosing characters in quotes.
- Python treats single quotes the same as double quotes.

Lists

- The most basic data structure in Python is the sequence.
- Each element of a sequence is assigned a number - its position or index.
- The first index is zero, the second index is one, and so forth.

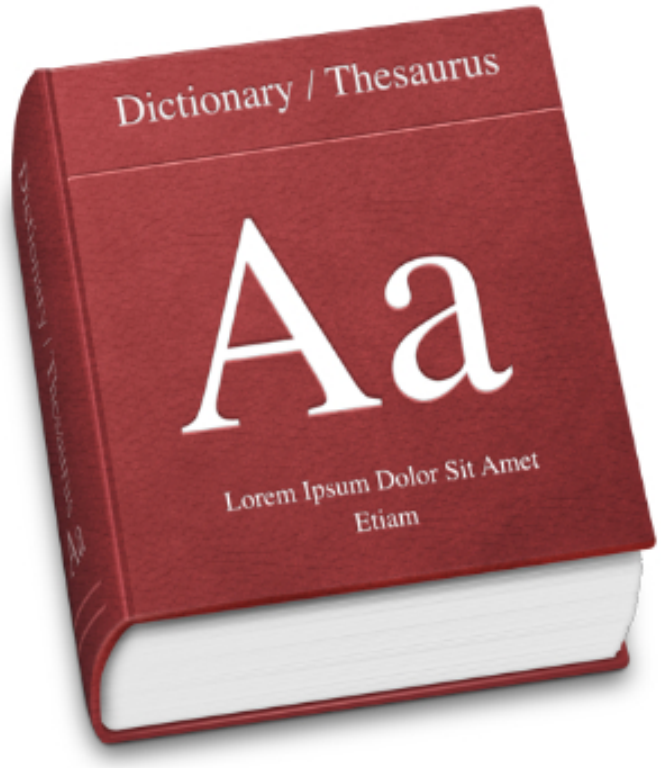
Tuples

- A tuple is a sequence of immutable Python objects.
- Tuples are sequences, just like lists.
- The differences between tuples and lists are:
  - the tuples cannot be changed unlike lists
  - tuples use parentheses, whereas lists use square brackets.



Dictionary

- Each key is separated from its value by a colon (:)
- The items are separated by commas
- The whole thing is enclosed in curly braces.

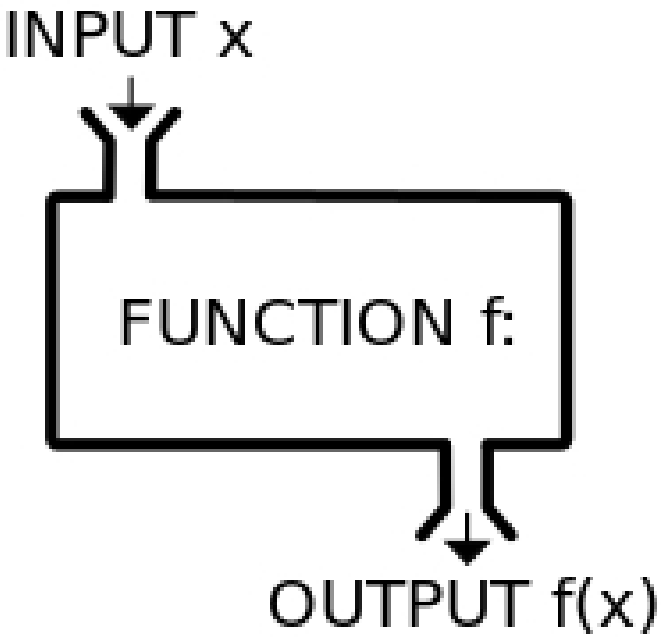


Flow Control

- If-Then-Else
- For
- While
- Exceptions

Functions

- Function blocks begin with the keyword def, followed by the function name and parentheses.
- Any input parameters or arguments should be placed within these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.



Functions

```
def square(x):
    return x * x
```

```
def hello():
    return "Hello"
```

```
def printme( str ):
    "This prints a passed string into this function"
    print str
    return
```

Classes

- The class statement creates a new class definition.
- The name of the class immediately follows the keyword class followed by a colon as follows

Classes

```
class Employee:
    """
    Common base class for all employees
    """
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount(self):
        print "Total Employee %d" % Employee.empCount

    def displayEmployee(self):
        print "Name : ", self.name,  ", Salary: ", self.salary
```

```
emp1 = Employee("Zara", 2000)
```

Garbage Collection

- Python deletes unneeded objects automatically to free the memory space.

Inheritance

- Instead of starting from scratch, you can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.

Inheritance

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
    """
    Optional class documentation string
    """
    # class_suite
```

Base Overloading Methods

- __init__(self [,args...]) : Constructor (with any optional arguments)
- __del__(self) : Destructor, deletes an object
- __repr__(self) : Evaluatable string representation
- __str__(self) : Printable string representation
- __lt__(self, other):
- __le__(self, other):
- __eq__(self, other):
- __ne__(self, other):
- __gt__(self, other):
- __ge__(self, other):
  These are the so-called rich comparison methods,
  and are called for comparison operators in preference to __cmp__() below.

Base Overloading Methods

- __cmp__ (self, x) : Called by comparison operations if rich comparison is not defined.
- __add__(self, other):
- __sub__(self, other):
- __mul__(self, other):
- __floordiv__(self, other):
- __mod__(self, other):
- __divmod__(self, other):
- __pow__(self, other[, modulo]):

Base Overloading Methods

- __lshift__(self, other):
- __rshift__(self, other):
- __and__(self, other):
- __xor__(self, other):
- __or__(self, other):
  These methods are called to implement the binary arithmetic operations +, -, *, //, %, divmod(), pow(), **, <<, >>, &, ^, |

Data Hiding

- An object's attributes may or may not be visible outside the class definition.
- You need to name attributes with a double underscore prefix, and those attributes then are not be directly visible to outsiders.
- Python protects those members by internally changing the name to include the class name.
- You can access such attributes as object._className__attrName

Data Hiding

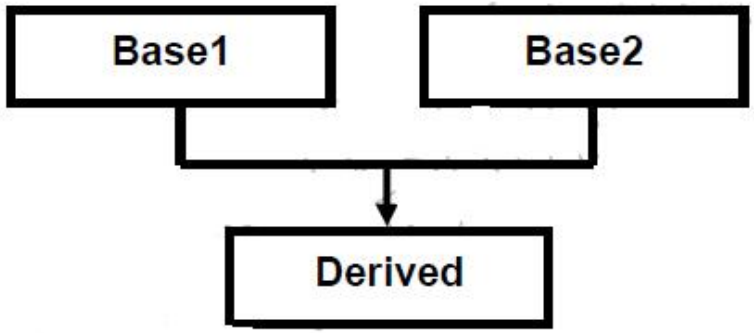
```
#!/usr/bin/python

class JustCounter:
    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print self.__secretCount

counter = JustCounter()
counter.count()
counter.count()
print counter.__secretCount
```

Data Hiding

```
1
2
Traceback (most recent call last):
File "test.py", line 12, in <module>
print counter.__secretCount
AttributeError: JustCounter instance has no attribute '__secretCount'
```

Multiple Inheritance

- Method Resolution Order (MRO)
- C3 Algorithm



C3 linearization

- The C3 superclass linearization is an algorithm used primarily to obtain the order in which methods should be inherited (the "linearization") in the presence of multiple inheritance, and is often termed "MRO" for Method Resolution Order.

C3 Algorithm

```
class O
class A extends O
class B extends O
class C extends O
class D extends O
class E extends O
class K1 extends A, B, C
class K2 extends D, B, E
class K3 extends D, A
class Z extends K1, K2, K3
```

C3 Algorithm

```
L(O)  := [O]

L(A)  := [A] + merge(L(O), [O])
= [A] + merge([O], [O])
= [A, O]

L(B)  := [B, O]
L(C)  := [C, O]
L(D)  := [D, O]
L(E)  := [E, O]

L(K1) := [K1] + merge(L(A), L(B), L(C), [A, B, C])
= [K1] + merge([A, O], [B, O], [C, O], [A, B, C])
= [K1, A] + merge([O], [B, O], [C, O], [B, C])
= [K1, A, B] + merge([O], [O], [C, O], [C])
= [K1, A, B, C] + merge([O], [O], [O])
= [K1, A, B, C, O]
```

C3 Algorithm

```
L(K2) := [K2] + merge(L(D), L(B), L(E), [D, B, E])
= [K2] + merge([D, O], [B, O], [E, O], [D, B, E])
= [K2, D] + merge([O], [B, O], [E, O], [B, E])
= [K2, D, B] + merge([O], [O], [E, O], [E])
= [K2, D, B, E] + merge([O], [O], [O])
= [K2, D, B, E, O]

L(K3) := [K3] + merge(L(D), L(A), [D, A])
= [K3] + merge([D, O], [A, O], [D, A])
= [K3, D] + merge([O], [A, O], [A])
= [K3, D, A] + merge([O], [O])
= [K3, D, A, O]
```

C3 Algorithm

```
L(Z)  := [Z] + merge(L(K1), L(K2), L(K3), [K1, K2, K3])
= [Z] + merge([K1, A, B, C, O], [K2, D, B, E, O], [K3, D, A, O], [K1, K2, K3])
= [Z, K1] + merge([A, B, C, O], [K2, D, B, E, O], [K3, D, A, O], [K2, K3])
= [Z, K1, K2] + merge([A, B, C, O], [D, B, E, O], [K3, D, A, O], [K3])
= [Z, K1, K2, K3] + merge([A, B, C, O], [D, B, E, O], [D, A, O])
= [Z, K1, K2, K3, D] + merge([A, B, C, O], [B, E, O], [A, O])
= [Z, K1, K2, K3, D, A] + merge([B, C, O], [B, E, O], [O])
= [Z, K1, K2, K3, D, A, B] + merge([C, O], [E, O], [O])
= [Z, K1, K2, K3, D, A, B, C] + merge([O], [E, O], [O])
= [Z, K1, K2, K3, D, A, B, C, E] + merge([O], [O], [O])
= [Z, K1, K2, K3, D, A, B, C, E, O]
```

Questions?