

An introduction to the Python Programming Language



Presentation Overview

- Introduction
- General Properties
- Variables and Data Types
- Strings and Numbers manipulation
- Control Flow
- Functions
- Classes and Objects
- Input and File I/O
- Useful Libraries



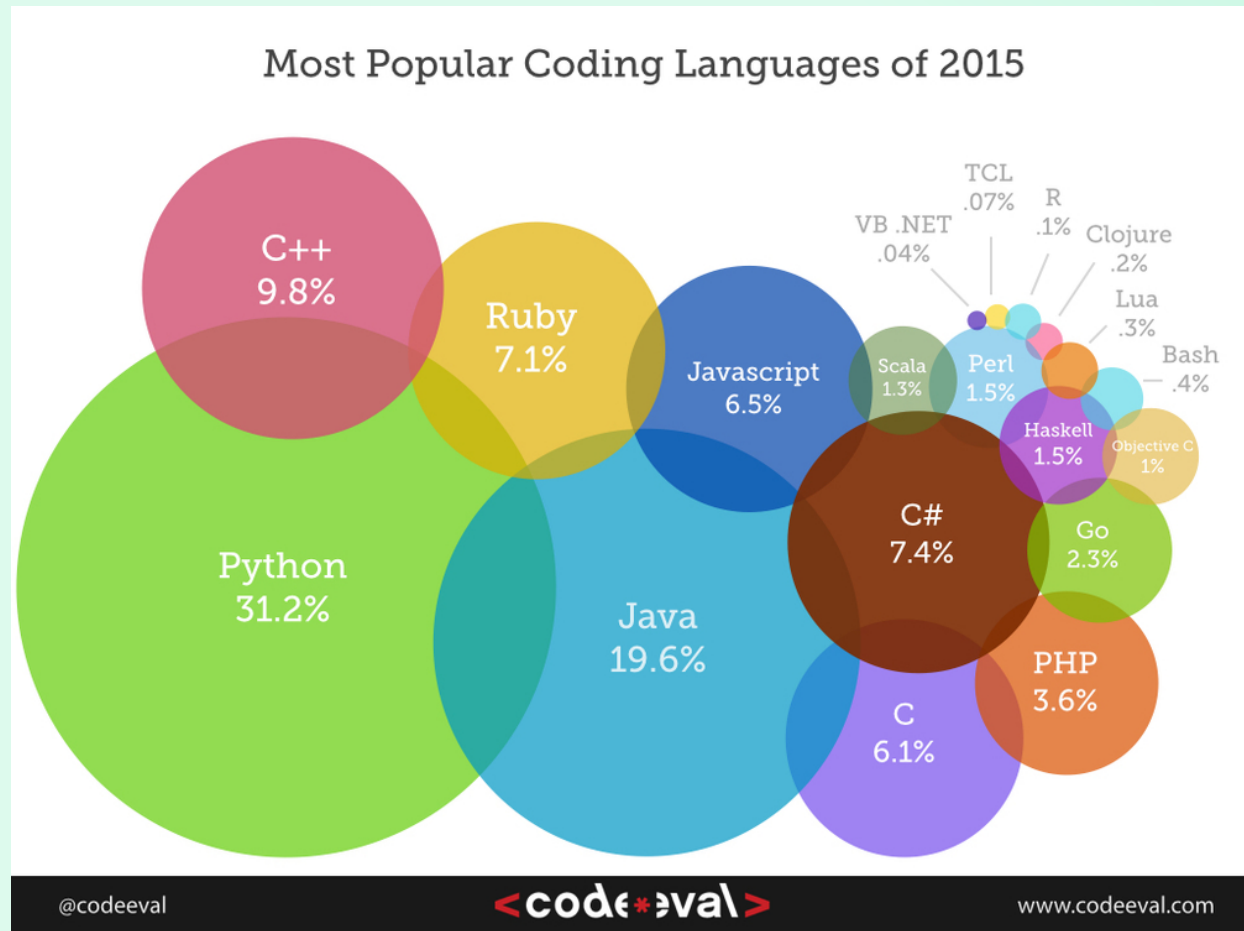
Introduction

- Importance of Python
- Versions of Python
- Interpreter
- First Program
- Development environment



Which of these languages do you know?

- C or C++
- Java
- Matlab
- JavaScript
- R
- Fortran





Python philosophy

- Coherence
 - not hard to read, write and maintain
- Power
- Scope
 - rapid development + large systems
- Objects
- Integration
 - hybrid systems
- Duck typing

Which Python version to use?



- **Main change:** Python 2's print statement has been replaced by the `print()` function
- Depends if you need third party libraries.



The Python Interpreter

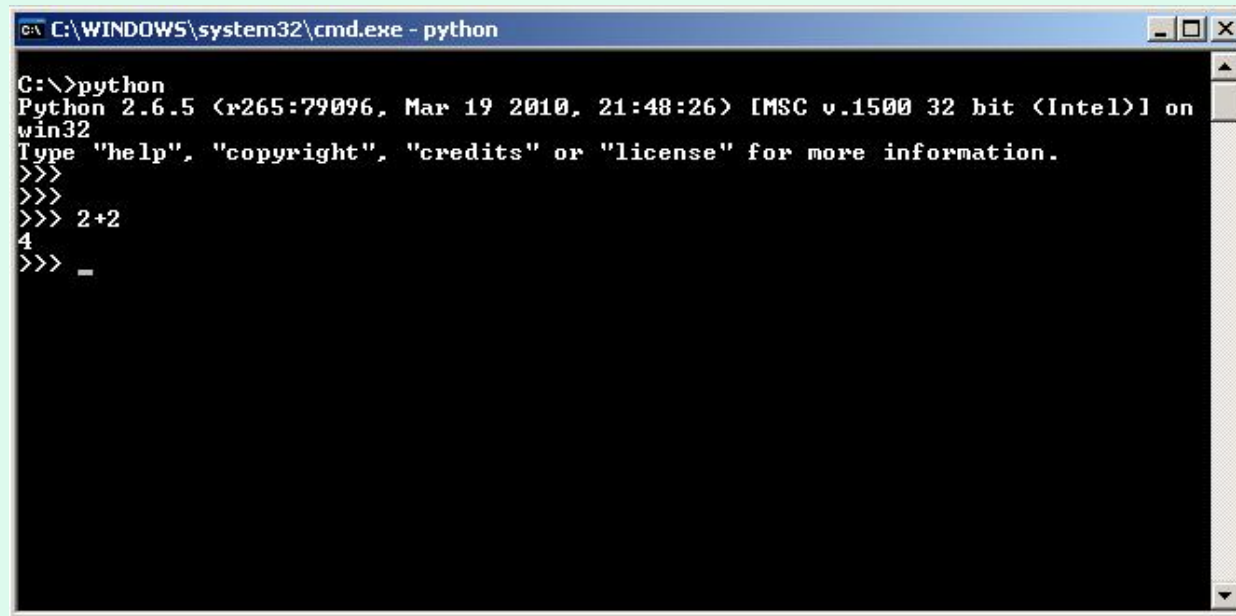
- Python is an interpreted language
- The interpreter provides an interactive environment to play with the language
- Results of expressions are printed on the screen

```
>>> 3 + 7
10
>>> 3 < 15
True
>>> 'print me'
'print me'
>>> print 'print me'
print me
>>>
```



Console

- Simplest way to execute python code
- Disadvantages:
 - Not reusable code
 - Very little support while coding



```
C:\WINDOWS\system32\cmd.exe - python
C:\>python
Python 2.6.5 <rt265:79096, Mar 19 2010, 21:48:26> [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>> 2+2
4
>>> _
```


Hello World program

As simple of...

```
print "Hello World"
```

Spyder Framework

The screenshot displays the Spyder Python IDE interface. The main window is titled "Editor - C:\Users\Steve\xy\fresnel\fresnel.py". The code editor shows the following Python code:

```
272 T = (s_data['T'] + p_data['T']) / 2.  
273 return {'R': R, 'T': T}  
274  
275 def position_resolved(layer, dist, fresnel_data):  
276     """  
277     Starting with output of fresnel_main(), calculate the Poynting vector  
278     and absorbed energy density a distance "dist" into layer number "layer"  
279     """  
280     vw = fresnel_data['vw_list'][layer]  
281     kz = fresnel_data['kz_list'][layer]  
282     th = fresnel_data['th_list'][layer]  
283     n = fresnel_data['n_list'][layer]  
284     n_0 = fresnel_data['n_list'][0]  
285     th_0 = fresnel_data['th_0']  
286     pol = fresnel_data['pol']  
287  
288     #amplitude of forward-moving wave is Ef, backwards is Eb  
289     Ef = vw[0] * exp(1j * kz * dist)  
290     Eb = vw[1] * exp(-1j * kz * dist)  
291  
292     #Poynting vector  
293     if(pol=='s'):  
294         poyn = ((n*cos(th)*conj(Ef+Eb)*(Ef-Eb)).real) / (n_0*cos(th_0)).real  
295     elif(pol=='p'):  
296         poyn = (((n*cos(th))*(Ef+Eb)*conj(Ef-Eb)).real)  
297             / (n_0*cos(th_0)).real  
298  
299     #absorbed energy density  
300     if(pol=='s'):  
301         absor = (n*cos(th)*kz*abs(Ef+Eb)**2).imag / (n_0*cos(th_0)).real  
302     elif(pol=='p'):  
303         absor = (n*cos(th))*  
304             (kz*abs(Ef-Eb)**2-conj(kz)*abs(Ef+Eb)**2)  
305             .imag / (n_0*cos(th_0)).real  
306     return({'poyn':poyn, 'absor':absor})  
307  
308 def find_in_structure(d_list,dist):  
309     """  
310     d_list is list of thicknesses of layers, all of which are finite.  
311  
312     dist is the distance from the front of the whole multilayer structure  
313     (i.e., from the start of layer 0.)  
314
```

The Object inspector on the right shows the function `fresnel_main(pol, n_list, d_list, th_0, lam_vac)` with its docstring. The console on the bottom right shows the following output:

```
In [8]: pv_sim.testtt()  
ISC = 4.103 mA/cm2  
EQE for 400-800nm = (4.103 mA/cm2) / (25.923 mA/cm2) = 15.8%  
Reflection into air = 16.2 mA/cm2 = 62.5%  
Absorption in mirror = 0.96 mA/cm2  
Thin-layer thicknesses in nm = [ 150.    70.    20.    20.    0.34]  
Absorption in thin layers = [ 1.18  0.51  4.64  1.91  0.    0.52]  
(for, respectively, [ITO,PEDOT,SubPC,C60,TPBi,graphene])  
C60 IQE = (1.49 mA/cm2) / (1.91 mA/cm2) = 77.8%  
SubPC IQE = (2.61 mA/cm2) / (4.64 mA/cm2) = 56.3%  
Out[8]: 4.1029296077801174  
  
In [9]: 1.18 + 0.51 + 4.64 + 1.91 + 0.52  
Out[9]: 8.76  
  
In [10]: 1.18 + 0.51 + 4.64 + 1.91 + 0.52 + 16.2 + 0.96  
Out[10]: 25.92  
  
In [11]:
```

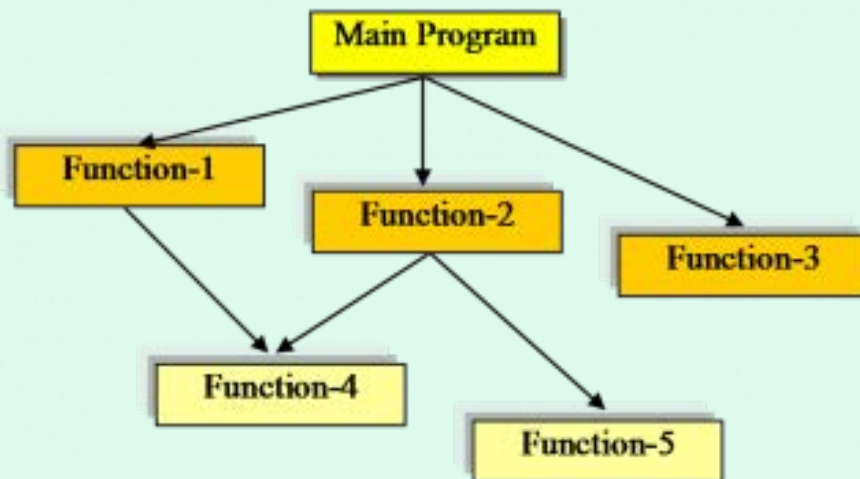
The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 289, Column: 28.

General properties

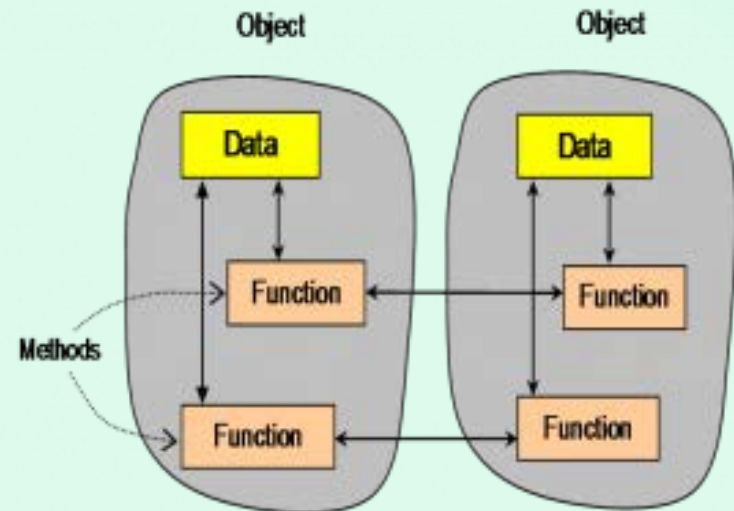
- Interpreted Language
- Programming Paradigms
 - Procedure Oriented Programing
 - Object Oriented Programming
- Python Structure
- Blocks of code and comments
- Keywords

POP vs OOP

Procedure-oriented Programming

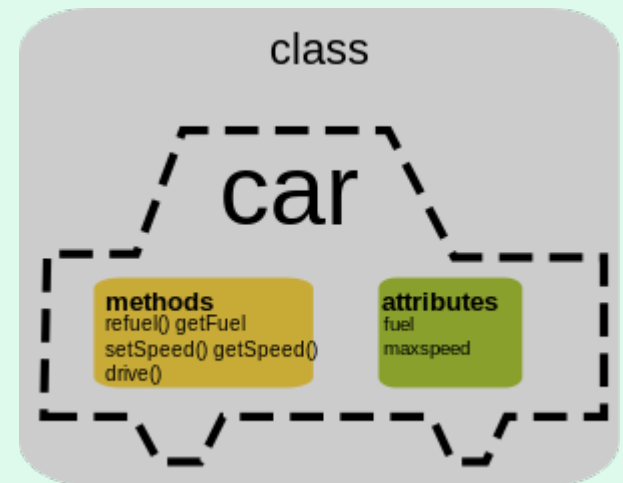
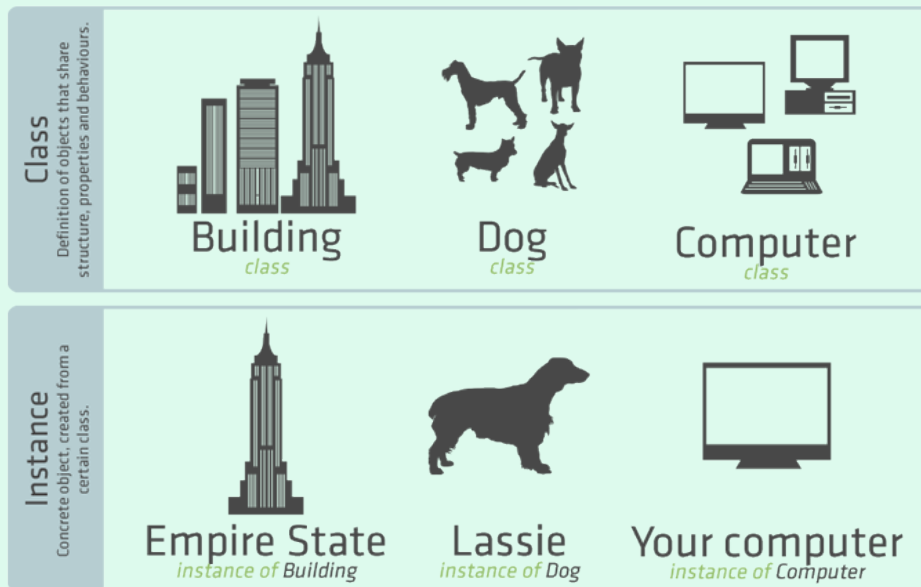


Object-oriented Programming



Object Oriented Programming

- Class: Template of the elements.
 - Formed by the definition of Methods and Attributes





The python structure

- modules: Python source files or C extensions
 - import, top-level via from, reload
- statements
 - create objects, functions, variables
 - control flow
- objects
 - everything is an object
 - automatically reclaimed when no longer needed



Modules

- Collection of functions and variables, typically in scripts.
- Each module has its own namespace
- Use the «**import**» statement to load them.
- File name is module name + .py
- Executed only when module is imported
- We can import only subsets of the module

Modules: Imports

import mymodule	Brings all elements of mymodule in, but must refer to as mymodule.<elem>
from mymodule import x	Imports x from mymodule right into this namespace
from mymodule import *	Imports all elements of mymodule into this namespace

No Braces {}

- Python uses *indentation* instead of braces to determine the scope of expressions.
- All lines must be indented the same amount to be part of the scope (or indented more if part of an inner scope)

```
def get_speed(space, time):  
    speed = space / time  
    return speed
```

```
number = -6  
if (number < 0):  
    print "Negative number"
```

Keywords

Keywords

- and
- del
- from
- not
- while
- as
- elif
- global
- or
- with
- assert
- else
- if
- pass
- yield
- break
- except
- import
- print
- class
- exec
- in
- raise
- continue
- finally
- is
- return
- def
- for
- lambda
- try



Variables

- Classical View
- Python View
- Numbers, Strings and Booleans
- Lists, Tuples and Dictionaries

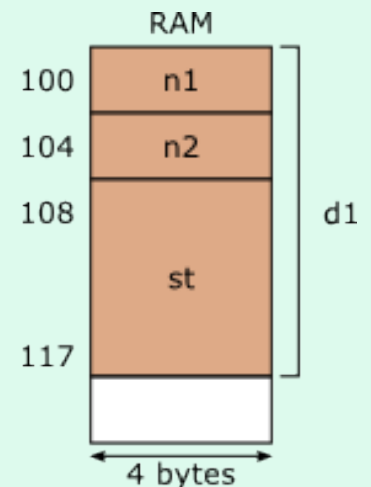
Classical View

A variable is just a container of information. It is a name that refers to a portion of memory.

In the classical view:

- A variables must be declared before being used.
 - It cannot change its type during the program.
 - Tell the OS that it exists and it reserves memory.
 - Different types: **int, float, char, bool**

```
struct contact
{
    int n1;
    float n2;
    char st[10];
} d1;
```



Variables in Python

- Are not declared, just assigned.
- The variable is created the first time you assign it a value.
- Type information is with the object
- Everything in Python is an object

Everything is an object

- Everything means everything, including functions and classes (more on this later!)
- Data type is a property of the object and not of the variable.

```
>>> x = 7
>>> x
7
>>> x = 'hello'
>>> x
'hello'
>>>
```

Numbers: Integers

- Integer – the equivalent of a C long
- Long Integer – an unbounded integer value.

```
>>> 132224
132224
>>> 132323 ** 2
17509376329L
>>>
```

Numbers: Floating Point

- **int**(x) converts x to an integer
- **float**(x) converts x to a floating point
- The interpreter shows a lot of digits

```
>>> 1.23232
1.232320000000000001
>>> print 1.23232
1.23232
>>> 1.3E7
13000000.0
>>> int(2.0)
2
>>> float(2)
2.0
```


Numbers are *immutable*

```
>>> x = 4.5
```

```
>>> y = x
```

```
>>> y += 3
```

```
>>> x
```

```
4.5
```

```
>>> y
```

```
7.5
```

x → 4.5

y → 4.5

x → 4.5

y → 7.5

Basic Mathematic Operators

Syntax	Description
<code>x + y</code>	Adds number <code>x</code> and number <code>y</code>
<code>x - y</code>	Subtracts <code>y</code> from <code>x</code>
<code>x * y</code>	Multiplies <code>x</code> by <code>y</code>
<code>x / y</code>	Divides <code>x</code> by <code>y</code> ; always produces a float (or a complex if <code>x</code> or <code>y</code> is complex) (Python 3 only)
<code>x // y</code>	Divides <code>x</code> by <code>y</code> ; truncates any fractional part so always produces an int result; see also the <code>round()</code> function
<code>x % y</code>	Produces the modulus (remainder) of dividing <code>x</code> by <code>y</code>
<code>x ** y</code>	Raises <code>x</code> to the power of <code>y</code> ; see also the <code>pow()</code> functions
<code>-x</code>	Negates <code>x</code> ; changes <code>x</code> 's sign if nonzero, does nothing if zero

- **math** and **numpy** library for advanced operations

String Literals

- Strings are *immutable*
- There is no char type like in C++ or Java
- Convert them to lists to operate at a char level.

```
>>> x = 'hello'  
>>> x = x + ' there'  
>>> x  
'hello there'
```

String Literals: Many Kinds

Can use single or double quotes, and three double quotes for a multi-line string

```
'I am a string'
```

```
"So am I!"
```

```
s = """And me too!  
though I am much longer  
than the others :) """
```

Substrings and Methods

```
>>> s = '012345'
>>> s[3]
'3'
>>> s[1:4]
'123'
>>> s[2:]
'2345'
>>> s[:4]
'0123'
>>> s[-2]
'4'
```

- **len**(String) – returns the number of characters in the String
- **str**(Object) – returns a String representation of the Object

```
>>> len(x)
6
>>> str(10.3)
'10.3'
```

Substrings and Methods

Name	Purpose
<code>len(s)</code>	Calculate the length of the string <code>s</code>
<code>+</code>	Add two strings together
<code>*</code>	Repeat a string
<code>s.find(x)</code>	Find the first position of <code>x</code> in the string <code>s</code>
<code>s.count(x)</code>	Count the number of times <code>x</code> is in the string <code>s</code>
<code>s.upper()</code> <code>s.lower()</code>	Return a new string that is all uppercase or lowercase
<code>s.replace(x, y)</code>	Return a new string that has replaced the substring <code>x</code> with the new substring <code>y</code>
<code>s.strip()</code>	Return a new string with whitespace stripped from the ends
<code>s.format()</code>	Format a string's contents

Booleans

- 0 and **None** are **False**
- Everything else is **True**
- **True** and **False** are aliases for 1 and 0 respectively

Boolean Expressions

- Compound boolean expressions short circuit
- **and** and **or** return one of the elements in the expression
- Note that when **None** is returned the interpreter does not print anything.

```
>>> True and False
False
>>> False or True
True
>>> 7 and 14
14
>>> None and 2
None
>>> None or 2
2
```


Logical Operators

Operation	Result
$x \mid y$	bitwise <i>or</i> of x and y
$x \wedge y$	bitwise <i>exclusive or</i> of x and y
$x \& y$	bitwise <i>and</i> of x and y
$x \ll n$	x shifted left by n bits
$x \gg n$	x shifted right by n bits
$\sim x$	the bits of x inverted

Lists

- Ordered collection of data
- Data can be of different types.
- Lists are *mutable*
- Issues with shared references and mutability
- Same subset operations as Strings

Lists

- lists can be heterogeneous
 - `a = ['spam', 'eggs', 100, 1234, 2*2]`
- Lists can be indexed and sliced:
 - `a[0]` → spam
 - `a[:2]` → ['spam', 'eggs']
- Lists can be manipulated
 - `a[2] = a[2] + 23`
 - `a[0:2] = [1, 12]`
 - `a[0:0] = []`
 - `len(a)` → 5

Lists methods

- `append(x)`
- `extend(L)`
 - append all items of list *L*. Also “+” operand
- `insert(i, x)`
- `remove(x)`
- `pop([i]), pop()`
 - create stack (FIFO), or queue (LIFO) → `pop(0)`
- `index(x)`
 - return the index for value *x*

Tuples

- Tuples are *immutable* versions of lists
- One strange point is the format to make a tuple with one element:
' ,' is needed to differentiate from the mathematical expression (2)

```
>>> x = (1,2,3)
>>> x[1:]
(2, 3)
>>> y = (2,)
>>> y
(2,)
```

Dictionaries

- A set of key-value pairs
- Dictionaries are *mutable*

```
>>> d = {1 : 'hello', 'two' : 42, 'blah' : [1,2,3]}  
>>> d  
{1: 'hello', 'two': 42, 'blah': [1, 2, 3]}  
>>> d['blah']  
[1, 2, 3]
```

Dictionaries: Add/Modify

- Entries can be changed by assigning to that entry

```
>>> d
{1: 'hello', 'two': 42, 'blah': [1, 2, 3]}
>>> d['two'] = 99
>>> d
{1: 'hello', 'two': 99, 'blah': [1, 2, 3]}
```

- Assigning to a key that does not exist adds an entry

```
>>> d[7] = 'new entry'
>>> d
{1: 'hello', 7: 'new entry', 'two': 99, 'blah': [1, 2, 3]}
```

Dictionaries: Deleting Elements

- The **del** method deletes an element from a dictionary

```
>>> d
{1: 'hello', 2: 'there', 10: 'world'}
>>> del(d[2])
>>> d
{1: 'hello', 10: 'world'}
```


Copying Dictionaries and Lists

- The built-in **list()** function will copy a list
- The dictionary has a method called **copy**
- For coping lists of lists, we use the function **copy.deepcopy()**

Data Type Summary

- Integers: 2323, 3234L
- Floating Point: 32.3, 3.1E2
- Lists: l = [1,2,3]
- Tuples: t = (1,2,3)
- Dictionaries: d = { 'hello' : 'there', 2 : 15 }

Data Type Summary

- Lists, Tuples, and Dictionaries can store any type (including other lists, tuples, and dictionaries!)
- Only lists and dictionaries are mutable
- All variables are references

Flow Control

- Instructions that are use to control de sequential execution of code.
 - Conditional Execution
 - if
 - else
 - Loops
 - for
 - while

If Statements

- The **if** statements allows us to execute a block of code only if a condition is fulfilled.
 - **if** (**condition**) :
 Block of code
- The condition is a boolean
 - True or False

```
age = 18
if (age < 16) :
    print "too young"
elif (age >= 35) :
    print "too old"
else :
    print "perfect match"
```

While Loops

- The **while** statements allows us to execute a block of code several times until a condition is fulfilled.
 - **While** (**condition**) :
Block of code
- The condition is a boolean
 - True or False

```
x = 1
while x < 10 :
    print x
    x = x + 1
```

Loop Control Statements

break	Jumps out of the closest enclosing loop
continue	Jumps to the top of the closest enclosing loop
pass	Does nothing, empty statement placeholder

For Loops

- Iterating through a list of values
 - Similar to Matlab and different to C or Java.

forloop1.py

```
for x in [1,7,13,2] :  
    print x
```

```
~: python forloop1.py  
1  
7  
13  
2
```

forloop2.py

```
for x in range(5) :  
    print x
```

```
~: python forloop2.py  
0  
1  
2  
3  
4
```

range(N) generates a list of numbers [0,1, ..., n-1]

Functions

- Function basics
- Input and Output Parameters manipulation
- Functions are objects



Function Basics

- A function is just a block of code (set of statements) that:
 - Perform operations over an input
 - Return an output
- A function cannot modify its input parameters
 - But it can modify the content of lists
 - Passing arguments by value or by reference
- Keyword `def` to create a function
 - `def func_name (parameters):`
 - Statements

```
def max(x,y) :  
    if x < y :  
        return x  
    else:  
        return y
```

Function Basics

- A function is just a block of code (set of statements) that:
 - Perform operations over an input
 - Return an output
- A function cannot modify its input parameters
 - But it can modify the content of lists
 - Passing arguments by value or by reference
- Keyword `def` to create a function
 - `def func_name (parameters):`
 - Statements

```
def max(x,y) :  
    if x < y :  
        return x  
    else:  
        return y
```

Parameters: Defaults

- Parameters can be assigned default values
- They are overridden if a parameter is given for them
- The type of the default doesn't limit the type of a parameter

```
>>> def foo(x = 3) :  
...     print x  
...  
>>> foo()  
3  
>>> foo(10)  
10  
>>> foo('hello')  
hello
```

Parameters: Named

- Call by name
- Any positional arguments must come before named ones in a call

```
>>> def foo (a,b,c) :  
...     print a, b, c  
...  
>>> foo(c = 10, a = 2, b = 14)  
2 14 10  
>>> foo(3, c = 2, b = 19)  
3 19 2
```

Functions are objects

- Can be assigned to a variable
- Can be passed as a parameter
- Can be returned from a function
- Functions are treated like any other variable in Python, the **def** statement simply assigns a function to a variable

Function names are like any variable

- Functions are objects
- The same reference rules hold for them as for other objects

```
>>> x = 10
>>> x
10
>>> def x () :
...     print 'hello'
>>> x
<function x at 0x619f0>
>>> x()
hello
>>> x = 'blah'
>>> x
'blah'
```

Functions as Parameters

```
def foo(f, a) :  
    return f(a)  
  
def bar(x) :  
    return x * x
```

funcasparam.py

```
>>> from funcasparam import *  
>>> foo(bar, 3)  
9
```

Note that the function **foo** takes two parameters and applies the first as a function with the second as its parameter

Functions Inside Functions

Since they are like any other object, you can have functions inside functions

```
def foo (x,y) :  
    def bar (z) :  
        return z * 2  
    return bar(x) + y
```

```
>>> from funcinfunc import *  
>>> foo(2,3)  
7
```

funcinfunc.py

Functions Returning Functions

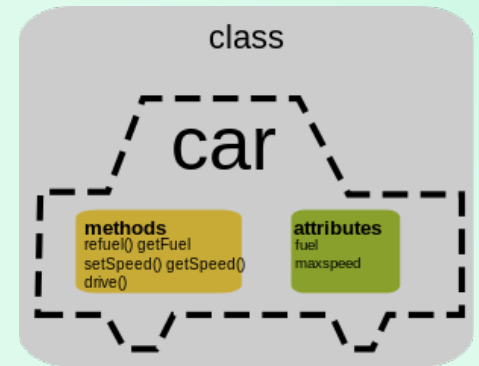
```
def foo (x) :  
    def bar(y) :  
        return x + y  
    return bar  
# main  
f = foo(3)  
print f  
print f(2)
```

funcreturnfunc.py

```
~: python funcreturnfunc.py  
<function bar at 0x612b0>  
5
```

Classes and Objects

- Class definition
- Examples
- Functions are objects



Class definitions

`Class ClassName (Base class):`

`<statement-1>`

`...`

`<statement-N>`

- must be executed
- can be executed conditionally
- creates new namespace

Example: Class creation

```
class Bag:
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def addtwice(self, x):
        self.add(x)
        self.add(x)
```

Example: Object instance

```
l = Bag()  
l.add('first')  
l.add_twice('second')  
  
print l.data  
['first', 'second', 'second']
```

Notes on classes

- Data attributes override method attributes with the same name.
- no real hiding → not usable to implement pure abstract data types
- clients (users) of an object can add data attributes
- first argument of method usually called **self**

Input and File Operations

- Prompt input
- Files in python
- Reading from files
- Writing into files

Prompt input

- The **raw_input**(string) method returns a line of user input as a string
- The parameter is used as a prompt
- The string can be converted by using the conversion methods **int**(string), **float**(string), etc.

Files in Python

- Manipulating files in python in 3 steps:
 - Opening the file with **open** (file_path, access_mode)
`archivo = open("./datos.txt","r")`
 - Reading or Writing the file
`datos_archivo = archivo.read()`
 - Closing the file with the **close** () attribute.
`archivo.close()`

Reading from files

<code>outfobj = open('data', 'r')</code>	Open the file 'data' for reading
<code>S = inflobj.read()</code>	Read whole file into one String
<code>S = inflobj.read(N)</code>	Reads N bytes (N >= 1)
<code>L = inflobj.readlines()</code>	Returns a list of line strings

Writing into files

<code>outflobj = open('data', 'w')</code>	Open the file 'data' for writing
<code>outflobj.write(S)</code>	Writes the string S to file
<code>outflobj.writelines(L)</code>	Writes each of the strings in list L to file
<code>outflobj.close()</code>	Closes the file

Access modes

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Libraries

➤ Numpy

➤ Matplotlib

➤ Pandas

THANK YOU !!