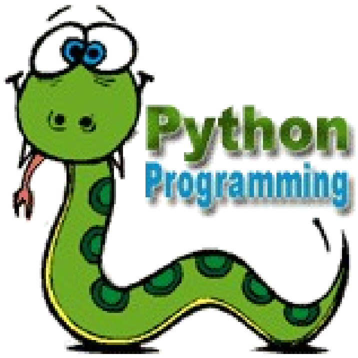
UNIVERSITÀ DI PADOVA



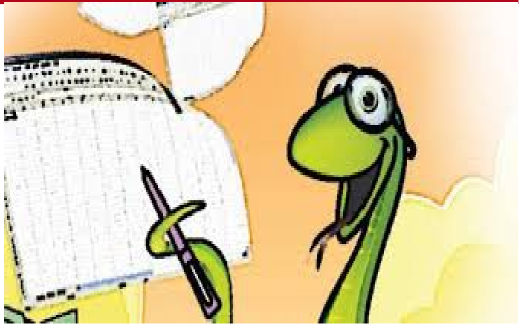
Prof. Mauro Conti

T.A. : Hossein Fereidooni & Moreno Ambrosin

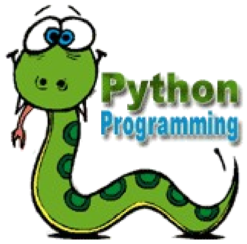
< 2014 April >

OUTLINE

What you will become familiar with during the Python programming course are as follows:

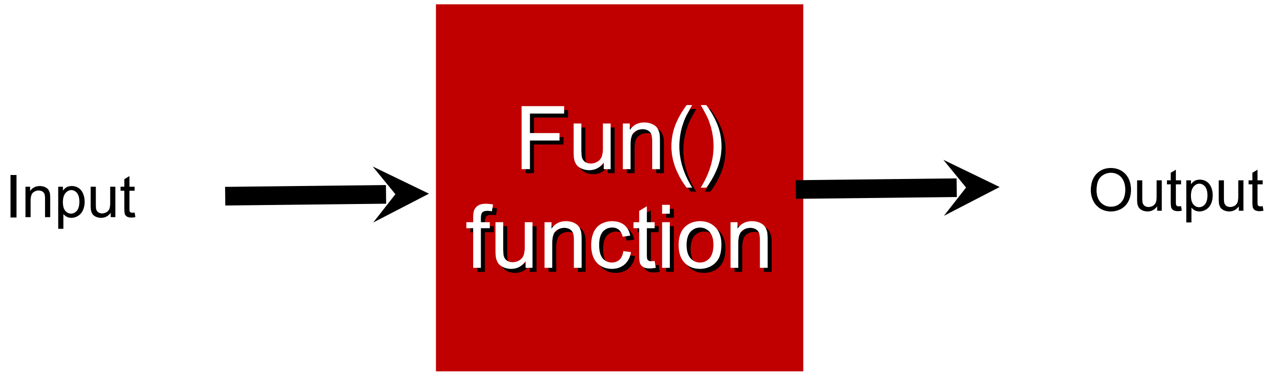


- Basic Operators
- Variable Types
- Numbers
- String
- Lists
- Tuples
- Dictionary
- Decision Making
- Loops
- Functions
- Modules
- Files I/O
- Exceptions
- Classes/Objects



To function or not to function...

- In Python a function is some reusable code that takes arguments(s) as input does some computation and then returns a result or results
- We define a function using the def reserved word
- We call/invoke the function by using the function name, parenthesis and arguments in an expression



Advantages of Function

- Reusable code
- Easier to maintain
- Increase readability
- Hide complexity from user

Function vs. Method

- Main difference between method and function
  - Function IS stand-alone block code (independent)
  - Method is a function that is bound to an abject

```
>>> randint(1,10)

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    randint(1,10)
NameError: name 'randint' is not defined
```

```
>>> import random
>>> print random.randint(1,10)
8
```

Python Functions

There are two kinds of functions in Python.

- Built-in functions that are provided as part of Python: raw_input(), type(), float(), int() ...
- Functions that we define ourselves and then use

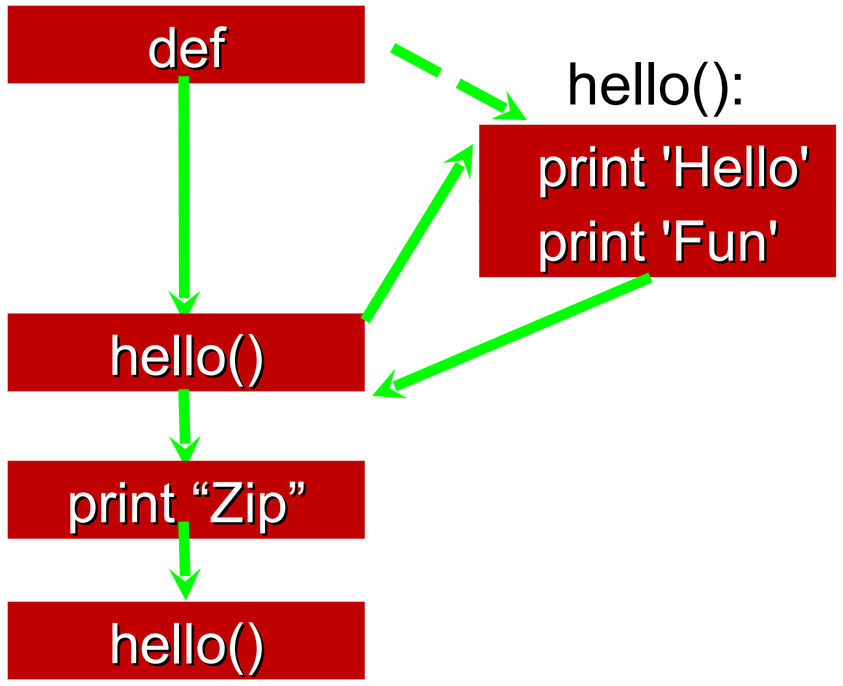
DEFINING A FUNCTION

- **Here are simple rules to define a function in Python:**

  - 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. You can also define parameters inside these parentheses.
  - The first statement of a function can be an optional statement - the documentation string of the function or docstring.
  - The code block within every function starts with a colon (:) and is indented.
  - The statement return [expression] exits a function, optionally passing back an expression to the caller.

```
>>> def uc(a):
            # uc() converts words into capital..
             a=a.upper()
             return(a)

>>> uc('hello Italia')
'HELLO ITALIA'
```

Stored (and reused) Steps



Program:

```
def thing():
    print 'Hello'
    print 'Fun'

thing()
print 'Zip'
thing()
```

Output:

```
Hello
Fun
Zip
Hello
Fun
```

We call these reusable pieces of code “functions”.

Building our Own Functions

- We create a new function using the def keyword followed by optional parameters in parenthesis.
- We indent the body of the function
- This defines the function but *does not* execute the body of the function

```
def print_lyrics():
    print "I'm a lumberjack, and I'm okay.”
    print 'I sleep all night and I work all day.'
```

```
x = 5
print 'Hello'

def print_lyrics():
    print "I'm a lumberjack, and I'm okay.”
    print 'I sleep all night and I work all day.'

print ‘Still,  No function'
x = x + 2
print x


print_lyrics():
```

```
Hello
Still, No function
7
```

print "I'm a lumberjack, and I'm okay."
print 'I sleep all night and I work all day.'

Definitions and Uses

- Once we have defined a function, we can call (or invoke) it as many times as we like .This is the store and reuse pattern

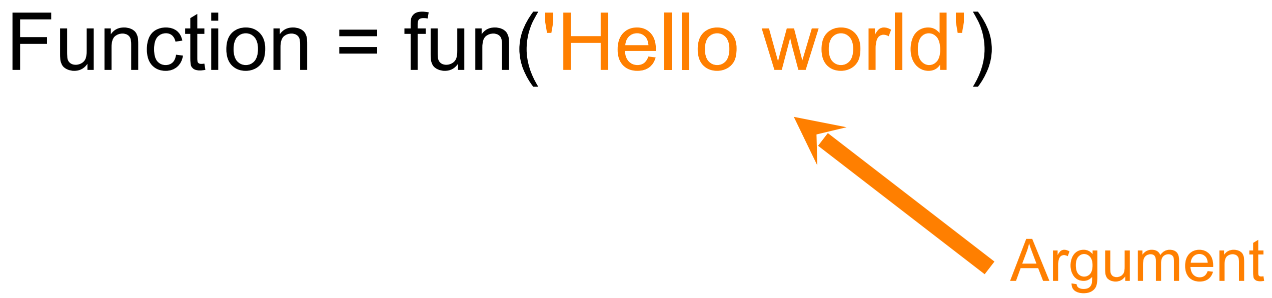
```
x = 5
print 'Hello'

def print_lyrics():
    print "I'm a lumberjack, and I'm okay.”
    print 'I sleep all night and I work all day.'

print ‘Still,  No function'
x = x + 2
print x


print_lyrics():
```

```
Hello
Still, No function
7
```

```
    print "I'm a lumberjack, and I'm okay."
      print 'I sleep all night and I work all
day.'
```

Arguments

- An argument is a value we pass into the function as its input when we call the function
- We use arguments so we can direct the function to do different kinds of work when we call it at different times
- We put the arguments in parenthesis after the name of the function



Parameters

- A parameter is a variable which we use in the function definition that allows the code in the function to access the arguments for a particular function invocation.

```
>>> def greet(lang):
...         if lang == 'es':
...             print 'Hola’
...         elif lang == 'fr':
...             print 'Bonjour’
...         else:
...             print 'Hello’
... >>> greet('en')
Hello
>>> greet('es')
Hola
>>> greet('fr')
Bonjour
```

Return Values

- Often a function will take its arguments, do some computation and return a value to be used as the value of the function call in the calling expression. The return keyword is used for this.

```
def greet():
    return "Hello”

print greet(), "Glenn”
print greet(), "Sally"
```

```
Hello Glenn
Hello Sally
```

Return Values

- A “fruitful” function is one that produces a result (or return value)
- The return statement ends the function execution and “sends back” the result of the function

```
>>> def greet(lang):
...         if lang == 'es':
...             return 'Hola’
...         elif lang == ‘it':
...             return ‘Ciao’
...         else:
...             return 'Hello’
```

```
... >>> print greet('en'),'Glenn’
Hello Glenn
>>> print greet('es'),'Sally’
Hola Sally
>>> print greet(‘it'),'Michael’
Ciao Michael
>>>
```

Values Return

```
>>> # return value , default value
def name(fn,ln='fereidooni'):
    fln=  'Your name is: %s %s ' % (fn,ln)
    return fln


>>> name('hossein')
'Your name is: hossein fereidooni ‘
 >>> name('hossein','conti')
'Your name is: hossein conti '
```

Void (non-fruitful) Functions

- When a function does not return a value, we call it a "void" function.
- Void functions are "not fruitful"
- Functions that return values are "fruitful" functions

Void Functions

```
def pause():
    raw_input("\n\nPress any key to continue...\n\n")

def quitMessage():
    print "Thank you for using this program"
    print "Goodbye"

def printThreeLines():
    for i in range(1,4):
        print 'this is line ' + str(i)

def printNineLines():
    for i in range(1,4):
        printThreeLines()
```

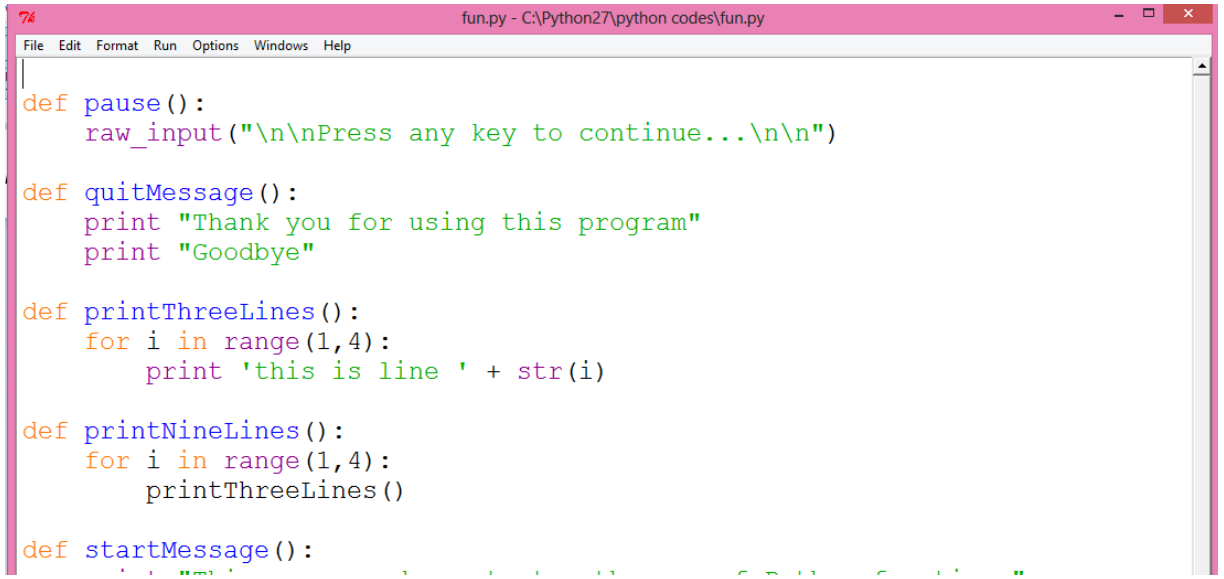
```
def startMessage():
    print "This program demonstrates the use of
Python functions"
    pause()

def blankLine():
    print

def clearScreen():
    for i in range(1,26):
        blankLine()
```

Importing and Modules

- Use functions defined in another file
- A Python module is a file with the same name (plus the *.py* extension)
- Formats of the command:
  - import somefile
  - from somefile import someFunction

import ...

```
>>> import func
>>> func.pause()


Press any key to continue...


>>> func.blankLine()


>>> func.quitMessage()
Thank you for using this program
Goodbye
```


fun.py - C:\Python27\python codes\fun.py
File Edit Format Run Options Windows Help

```
def pause():
    raw_input("\n\nPress any key to continue...\n\n")

def quitMessage():
    print "Thank you for using this program"
    print "Goodbye"

def printThreeLines():
    for i in range(1,4):
        print 'this is line ' + str(i)

def printNineLines():
    for i in range(1,4):
        printThreeLines()

def startMessage():
```

- Everything in fun.py gets imported.

from … import *

```
>>> from func import pause
>>> pause()


Press any key to continue...


>>> blankList()

Traceback (most recent call last):
  File "<pyshell#44>", line 1, in <module>
    blankList()
NameError: name 'blankList' is not defined
```

```
>>> quitMessage()

Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    quitMessage()
NameError: name 'quitMessage' is not defined
```

- Only the item *pause* in func.py gets imported.

Directories for module files

- Where does Python look for module files?
- The list of directories where Python will look for the files to be imported is sys.path

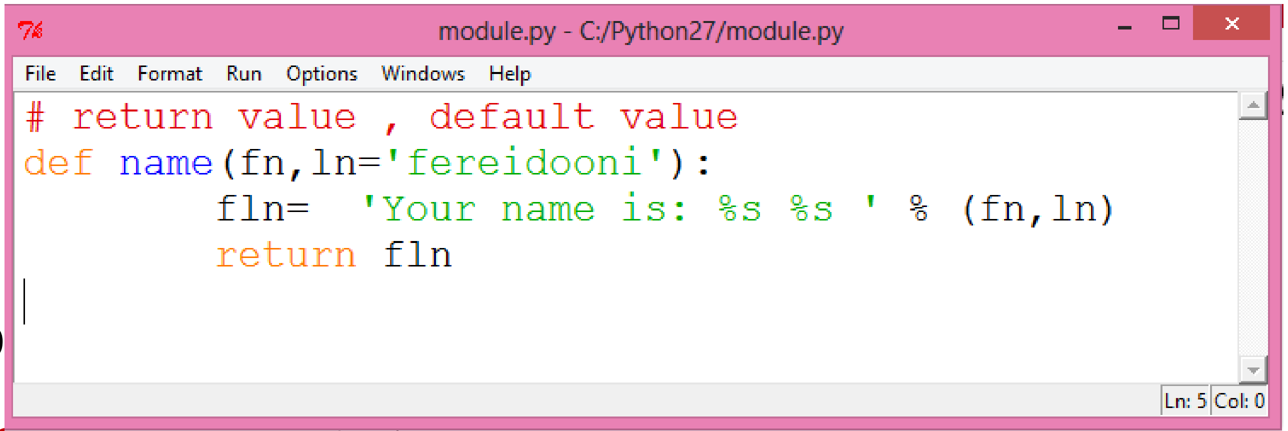
```
>>> import sys

>>> sys.path
```

- To add a directory of your own to this list, append it to this list

```
sys.path.append("C:\\/my/new/path" )
```

Example of import

- Save your file with .py extension (module.py)
- Import module
- Call module.functionName()

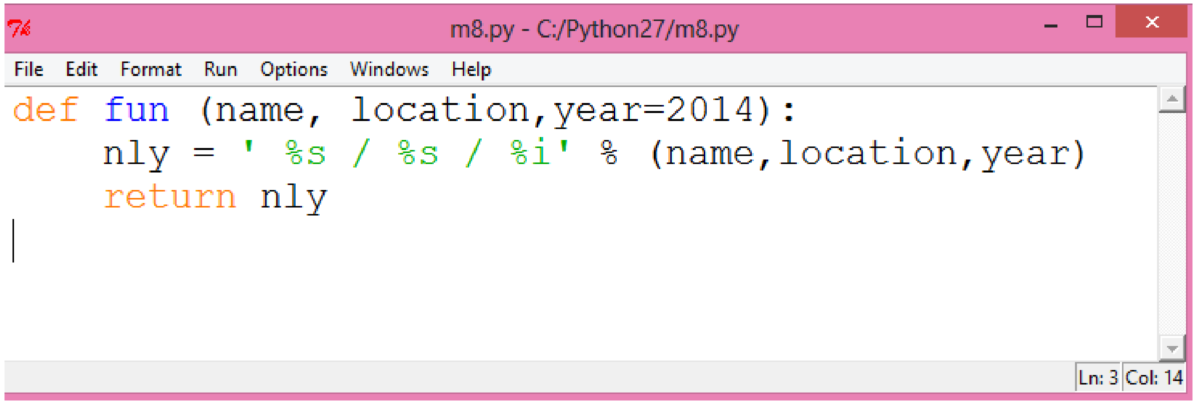

module.py - C:/Python27/module.py
File Edit Format Run Options Windows Help
Ln: 5 Col: 0

```
# return value , default value
def name(fn,ln='fereidooni'):
        fln=  'Your name is: %s %s ' % (fn,ln)
        return fln
```

```
import module
>>> module.name('ho
'Your name is: hossein fereidooni
```

Example of import

```
>>> import m8
>>> m8.fun('hossein','Padova',2014)
' hossein / Padova / 2014'
>>> m8.fun('Mauro','Roma')
' Mauro / Roma / 2014'
```


m8.py - C:/Python27/m8.py
File Edit Format Run Options Windows Help
Ln: 3 Col: 14

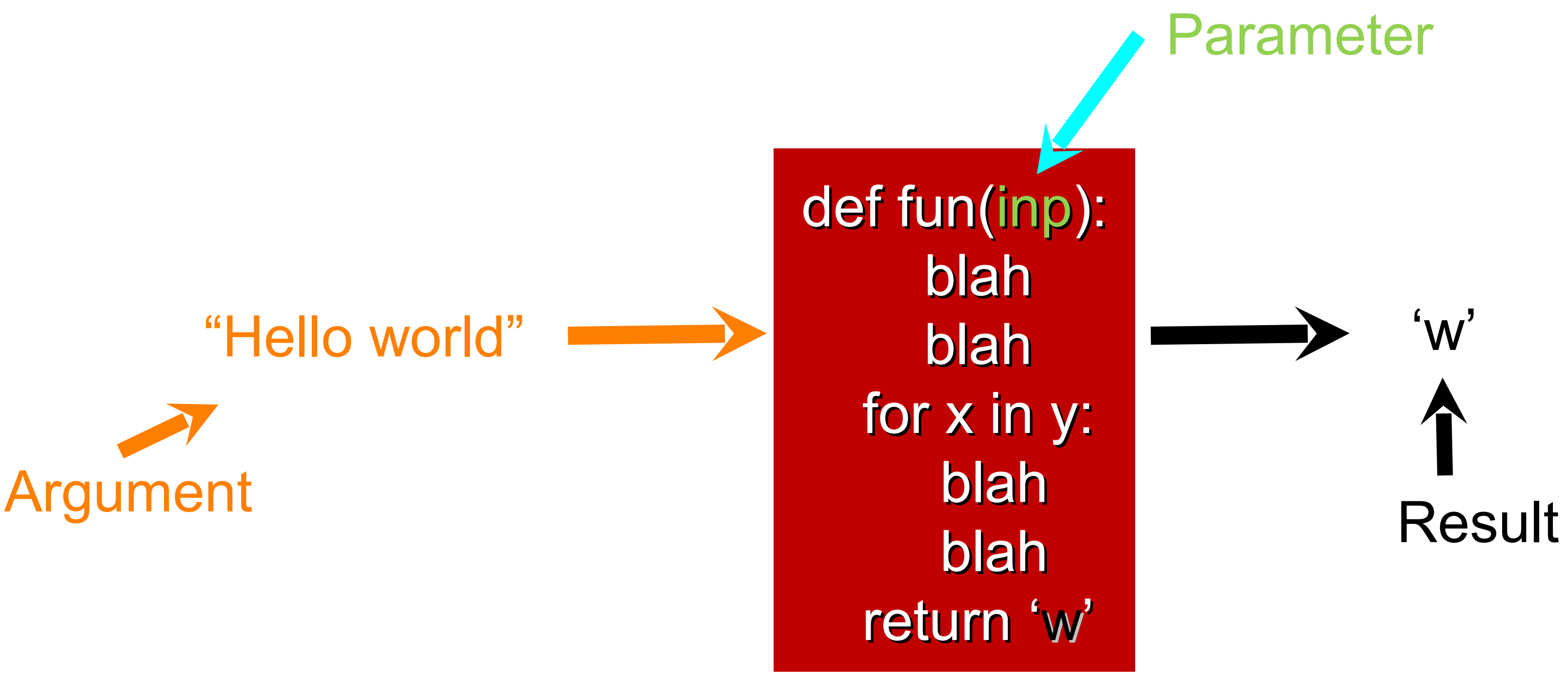
```
def fun (name, location,year=2014):
    nly = ' %s / %s / %i' % (name,location,year)
    return nly
```

Calling Function

```
def fun (name, location,year=2014):
    nly = ' %s / %s / %i' % (name,location,year)
    return nly

>>> fun('moreno','Padova',2013)
' moreno / Padova / 2013'
```

Arguments, Parameters, and Results

>>> Function = fun ('Hello world')



No Return Values

```
def typer(x) :
    if type(x)==int:
      print " Input is an integer"
    elif type(x)== str:
      print ' Input is a string'
    else:
      print 'this is neither an integer nor string'
```

```
>>> typer('hi')
 Input is a string
>>> typer(3)
 Input is an integer
>>> typer(3.5)
this is neither an integer nor string

>>> import m8
>>> m8.typer(10)
 Input is an integer
```

Multiple Parameters / Arguments

- We can define more than one parameter in the function definition
- We simply add more arguments when we call the function
- We *match the number and order* of arguments and parameters

```
def addtwo(a, b):
    added = a + b
    return added
x = addtwo(3, 5)
print x
```

Multiple Function Argument

```
def one(a,*arg):
    print 'this is mandatory parameter: ',a
    print 'these are extra parameters: ',arg
```

```
>>> one(1,2,3,4)
this is mandatory parameter:  1
these are extra parameters:  (2, 3, 4)
>>> one(1)
this is mandatory parameter:  1
these are extra parameters:  ()
>>> one(1,2,3,4,5,6,7,9)
this is mandatory parameter:  1
these are extra parameters:  (2, 3, 4, 5, 6, 7, 9)
```

Multiple Function Argument

```
def foo(first, second, third, *therest):
    print "First: %s" % first
    print "Second: %s" % second
    print "Third: %s" % third
    print "And all the rest... %s" % list(therest)
```

Now, calling the foo

```
>>> foo(1,2,3,4,5,6)
First: 1
Second: 2
Third: 3
And all the rest... [4, 5, 6]
```

```
>>> foo (1,2,3,4,5,6,7,'hi')
First: 1
Second: 2
Third: 3
And all the rest... [4, 5, 6, 7, 'hi']
```

The "therest" variable is a list of variables

File Processing

- A text file can be thought of as a sequence of lines

```
LOCUS       NC_005213            490885 bp    DNA     circular CON 10-JUN-2013
DEFINITION  Nanoarchaeum equitans Kin4-M chromosome, complete genome.
ACCESSION   NC_005213
VERSION     NC_005213.1  GI:38349555
DBLINK      Project: 58009
            BioProject: PRJNA58009
KEYWORDS    .
SOURCE      Nanoarchaeum equitans Kin4-M
  ORGANISM  Nanoarchaeum equitans Kin4-M
            Archaea; Nanoarchaeota; Nanoarchaeum.
REFERENCE   1  (bases 1 to 490885)
  AUTHORS   Waters,E., Hohn,M.J., Ahel,I., Graham,D.E., Adams,M.D.
…………………………………………
```

Opening a File

- Before we can read the contents of the file we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the open() function
- open() returns a “file handle” - a variable used to perform operations on the file
- Like: “File -> Open” in a Word Processor

Using open() and close()

- handle = open(filename, mode)
  - returns a handle use to manipulate the file
  - filename is a string
  - mode is optional and should be 'r' if we are planning reading the file and 'w' if we are going to write to the file.

```
# Open a file
fo = open(‘foo.txt', 'r')
# Close opend file
fo.close()
```

Using write()

```
# Open a file
fo = open("foo.txt", "wb")
# Write inside the file
fo.write('this is a test to write something in the file')
```

Using read()

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(15);
print "Read String is : ", str
print len(str)
# Close opend file
fo.close()
```

- We can read the whole file into a single string.

File as a Sequence

- A opened file for reading can be treated as a sequence of strings where *each line in the file is a string in the sequence*
- We can use the for statement to iterate through a sequence
- Remember - a sequence is an ordered set

```
fo = open(‘foo.txt')
for i in fo:
  print i
```

Counting Lines in a File

```
fo =  open(foo.txt')
count = 0
for line in fo:
 count = count + 1
print 'Line Count:', count
```

- Open a file read-only
- Use a for loop to read each line
- Count the lines and print out the number of lines

Searching Through a File

```
fo = open('foo.txt')
for line in fo:
  if line.startswith('I'):
     print line
```

- We can put an if statement in our for loop to only print lines that meet some criteria

REFERENCES

1. http://www.tutorialspoint.com/index.htm
2. http://docs.python.org/lib/string-methods.html
3. http://www.pythonlearn.com/

Contact

- **Website: http://www.math.unipd.it/~hossein/fereidooni.htm**
- **E-mail: hossein@math.unipd.it**
- **Skype: fereidooni1983**