PYTHON WORKSHOP

BCBGSO

Yuan Wang

Why Programming?

- The power of automation, the ability to think in algorithmic ways, a new approach to solve problems.

Why Python?

- Easy to learn and understand
- Natural and forgiving syntax
- Broad and Powerful
- Might be slower compared with C++ or Java

Python 2 or Python 3?

- All materials covered in this workshop will be written in Python 2.
- Python 2.6 and 2.7 are still very robust and widely used, also some of the less disruptive improvements in 3.0 and 3.1 have been backported to 2.6 and 2.7.
- Default versions on most Linux and Mac machines are Python 2
- Python 2 and 3 don't vary very much when you first begin to learn Python

Where to get help with Python?

- http://www.python.org

  Tutorial

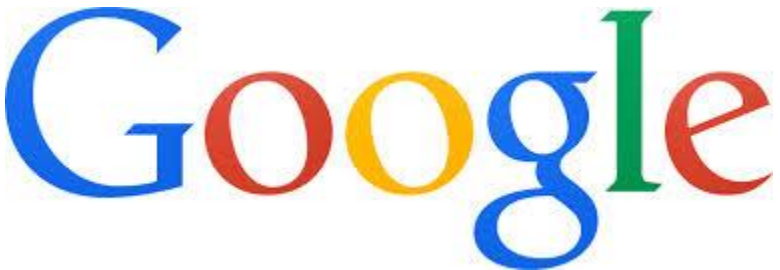
  Library Reference

  Language Reference

- https://wiki.python.org/moin/BeginnersGuide/Programmers

List of more than enough tutorials (non-interactive and interactive) and video tutorials

- Always google



Getting Started with Python

- To start Python, open a command prompt or a terminal, then type:

  python

- To exit from Python, simply type:

  exit() or quit()

  then hit ‘enter’

Navigate to your workshop folder

- Windows:

  open command prompt

  `cd Desktop\PythonWorkshop`

  `dir` or `cd` to make sure

- Linux + Mac OS:

  open a terminal

  `cd Desktop/PythonWorkshop`

  `ls` or `pwd` to make sure

Running Python code from file -- “Hello World”

- In a command prompt or terminal, navigate to the folder where your .py file is located, type:

  python PretendForAMomentImARealFile.py

- Open your system’s text editor, type in:

  print ‘Hello World’

  Save in your Workshop folder as HelloWorld.py

- Navigate to your Workshop folder, type:

  python HelloWorld.py

Python syntax

- Indentation: Python uses indentation to indicate blocks of code – the number of spaces in indentation for all statements within the same block needs to be the same.
- Comments: any line starts with # or any block of code surrounded by ‘’’ (3 single quotes) will not be executed
- Case-sensitive

Basic Python Variables

- Variable – Reserved memory locations to store values.
- Integers – Same precision as a C long, usually a 32-bit binary number
- Floats – Number with a decimal point, implemented as a C double
- Strings – A contiguous set of characters in between quotation marks

Numeric Operators

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
**	Exponent - Performs exponential (power) calculation on operators
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.

Code Example

- We have two variables: a and b, a = 5, b = 3

  What are the values of the following:

  a + b a – b a * b

  a / b a % b a ** b

  a // b

- Navigate to your Python Workshop folder, and run *NumOperation.py*

  python NumOperation.py

Integers and Floats

- If you choose to leave out the fractional portion of a set of numbers, Python will always return it as an integer
- If you put in the fractional part , even if it is just on one number and it is a .0, Python will always return it as a float.

Strings in Python

- Strings – A contiguous set of characters in between quotation marks

s = ‘tofu is better than planned economy’

s = “tofu is better than planned economy”

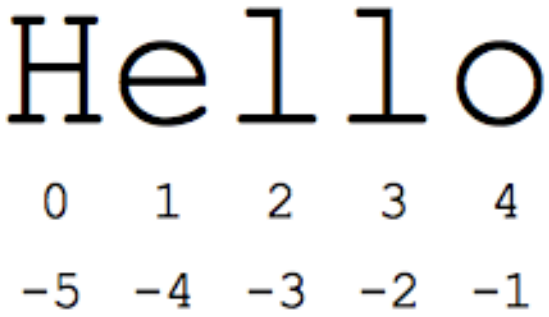
both single quote and double quote are OK

s = “I didn’t do it”

s = ‘double “quote”’

String Indexing

- Characters can be accessed using standard [] syntax, index starts from 0



s = ‘tofu is better than planned economy’
s[0] = ‘t’
s[-2] = ‘m’
s[5:11] = ‘is bet’
s[:2] = ‘to’
s[2:] = ‘fu is better than planned economy’

Strings Methods

- s.lower(), s.upper() : return lowercase or uppercase version of the string
- s.strip() : return a string with whitespace removed from the start and the end
- s.startswith(‘other’), s.endswith(‘other’): tests if the string starts or ends with a given string
- s.find('other') -- searches for the given other string (not a regular expression) within s, and returns the first index where it begins or -1 if not found

Strings Methods

- s.replace('old', 'new') -- returns a string where all occurrences of 'old' have been replaced by 'new'
- s.split('delim') -- returns a list of substrings separated by the given delimiter. The delimiter is not a regular expression, it's just text.
- To concatenate two strings: use ‘+’

  s1 = ‘Hello’; s2 = ‘World’

  s1 + s2 = ‘HelloWorld’

int(), float(), and str()

- int(x): convert a number or string x to an integer, if possible
- float(x): convert a string or number x to a floating point number, if possible
- str(obj): return a nice string representation of the object.

String Formatting

- print ‘%s is the %dth day of %d’ %(’today’,30,2015)

Format Symbol	Conversion
%s	String
%d	Integer
%f	Float
%e	Exponential notation

Exercise 1 – open file ex1.py

- 1. Print the sequence alone (without header '6404|')
- 2. Print the sequence alone, all in lower-case
- 3. Print the index of substring 'GAAC' (first occurrence only)
- 4. Make it a RNA sequence and print it out (change every T to U)
- 5. Add 'CAGACACGC' to the end of the sequence and print it out

Lists in Python

- A list of comma-separated values (items) between square brackets
- Items in a list do NOT have to be the same type
- MyList = [6828, ‘cui’, 11.16]
- MyList[0] = 6828
- MyList[1:3] = [‘cui’, 11.16]
- MyNewList = [6828, ‘cui’, 11.16, [1, 2, 3]]
- MyNewList[2:4] = [11.16, [1,2,3]]

Updating Lists

- Update of a list element can be done using its index.

  MyList[0] = 5717

- A new element can be added to the end of the list using append() or extend()

  MyList.append('relax')

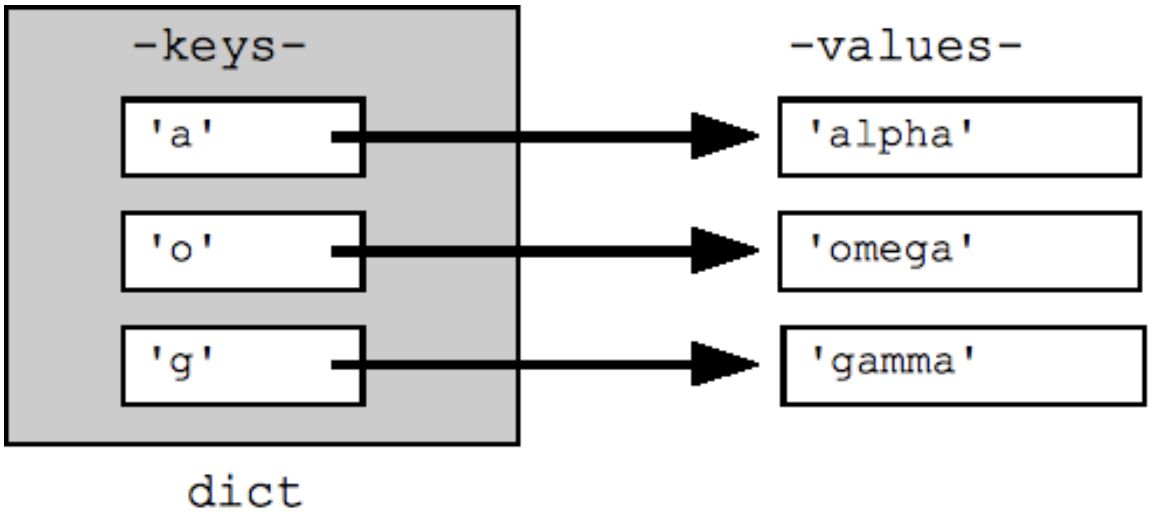
  MyList.extend('how')

List Methods

- list.count(obj): returns how many times object obj occurs in list
- list.index(obj): returns the lowest index in list that obj occurs
- list.insert([index,obj]): insert obj into list at its offset index
- list.remove(obj): remove the first obj occurs in list
- list.reverse(): reverse objects in list in place

Python Dictionaries

- A dictionary is mutable and is another container type that can store any number of Python objects, including other container types. Dictionaries consist of pairs of keys and their corresponding values.
- d = {‘a’ : ‘alpha’, ‘o’ : ‘omega’, ‘g’ : ‘gamma’}



Updating Dictionaries

- d = {‘a’ : ‘alpha’, ‘o’ : ‘omega’, ‘g’ : ‘gamma’}
- d[‘o’] = ‘omg’ #update existing entry
- d[‘b’] = ‘beta’ #create new entry
- del d[‘a’] #remove an existing entry
- d.clear() #remove all entries from d
- d.update(d2) #add all of d2’s entries to d
- len(d): this would return the number of entries in d

Exercise 2 – open file ex2.py

- 1. Add following entries into the CodonTable:

  GGG : G

  UAA : STOP
- 2. Print the number of entries in CodonTable, which should be 64
- 3. Add 'AAC' and 'GAA' to RNAs
- 4. Translate index 0, 2, 5 in RNAs into amino acids (using CodonTable), print the results out, respectively

Boolean Values

- True, False are the Boolean values in Python (note the capitalized ‘T’ and ‘F’)

- numeric zero, None, the empty string, an empty list, an empty dictionary, any empty container are all considered as False, all other values are considered True

Logical Operators

Operator	Description
==	Equal to
!=	Not Equal to
> (<)	Greater (Less than)
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in	Membership tests

True or False?

- a = 3; b = 5; c = [‘Ala’, ‘Nonpolar’, 7];
- a > b — False
- a != b — True
- a == 3 — True
- (a > b) or (a < b) — True
- (a >= 3) and (b < 2) — False
- not (a == 3) — False
- a in c — False
- b not in c — True
- not ((a != 3) or (b < -2) or (b in c) or a) — False

“If” Statement in Python

```
if condition1:
        statements
elif condition2:
        statements
elif condition3:
        statements
else:
        statements
```

Special Attention to the colons and indentations

“If” Statement Example

```
lunch = ['salad','burger','steak fries and cheesecake']
myChoice = lunch[0]

if myChoice == 'salad':
    print 'Total calories: %d' % 225
elif myChoice == 'burger':
    print 'Total calories: %d' % 335
elif myChoice.startswith('ste'):
    print 'Go to gym, now'
else:
    print 'Not sure about what you had'
```

“For” Loops

- *For* loops are traditionally used when you have a piece of code which you want to repeat *n* number of times.

```
for x in y:
        code blocks
```

- Strings, lists, dictionaries are all iterable

“For” Loop Example

```
for i in range(0,10):
    print i
```

```
for i in range(0,10,2):
    print i
```

python™

“While” Loop

- Unlike “for” loop, “while” loop is used when a condition is to be met, or if you want a piece of code to repeat forever.

```
While condition:
        code blocks
```

“While” Loop Example

```
i = 0
while i <= 10:
    print i
    i = i + 1
```

python™

“For” or “While”?

- **for** statement iterates through a collection or iterable object or generator function (Strings, Lists, Dictionaries, Files, etc.), usually used when the number of iterations needed is known.
- **while** statement is usually used when you don’t have a clean data structure, so that iteration can be controlled when certain condition becomes false.

“Break” and “Continue”

- break: terminates the current loop and resumes execution from the next statement out of the loop. Can be used in both for and while loops.
- continue: returns the control to the beginning of the loop. This rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

Example – Break and Continue

```
for i in range(0,10):
    if i >= 8:
        break
    else:
        print I
```

```
for i in range(0,10):
    if i % 2 == 0:
        continue
    else:
        print i
```

python™

Exercise 3 – open file ex3.py

- 1. Count the number of 'C's in the sequence and print it out
- 2. Calculate its 'GC-content' and print it out

  GC-content = (#G + #C) / sequence length
- 3. Print out its conplementary chain sequence

  (A to T, T to A, C to G, G to T)
- 4. * Transcribe the sequence to mRNA (T to U) and then translate it to protein

  ** Stop translating at 'STOP' codon or

  ** Neglect all 'STOP' codons

Functions

- A block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

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 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. A return statement with no arguments is the same as return None.

Example

```
def areaCalculator(a,b):
    area = a * b
    return area

print areaCalculator(3,4)
```

Files I/O

- To open a file:

  f = open(file_name [, access_mode])

  f = open('test.txt','r+') will open file 'test.txt' for both reading and writing

Access Mode	Description
r	Reading only
r+	Reading and writing
w	Writing only
w+	Reading and writing
a	Appending only
a+	Appending and reading

Read from file once file is open

- f.read([count]): parameter ‘count’ is the number of bytes to be read from the beginning of the file, if count is missing, it will read the whole file
- f.readline([count]): this will read one line from file each time it is called, parameter count acts similarly as above.
- file object itself is iterable, the following block is very commonly used:

```
f = open(‘test.txt’,’r’)
for line in f:
    line.strip()
    ……
```

Write to file

- f = open(‘test.txt’, ’w’)
- f.write(‘this will be written to the file\n’)

  f.write(formatted strings)
- f.close()

  this will close the file, this is not necessary as Python will close the file automatically, this is just good practice

Class and Object-oriented Programming

- Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation.

Good Coding Practices

- Keep commenting: it makes code maintenance so much easier
- Follow naming conventions
- If your code is going to be once: write codes fast, even the run time might be longer
- If your code is going to be used repeatedly: devote more time, look for better and efficient algorithms

Exercise 4 – open file ex4.py

- 1. Write a function that returns its GC-content when a sequence is given

  GC_Content = (#G + #C) / length(seq)

- 2. SeqForEX4.txt contains 5 different DNA sequences with its corresponding headers, find a way to read in the 5 sequences alone and print them out, respectively
- 3. Calculate the GC-content for all 5 sequences

- * Find the sequence with the highest GC-content, write its ID from header and its coresponding GC-content to a file named 'result.txt'