GRADUATE LINGUISTICS STUDENT ASSOCIATION PYTHON TUTORIAL

Led by Edwin Ko & James Maguire

WHAT IS COMPUTER SCIENCE?

- Science of computer?
- What can be computed?
  - Computers run entirely on computations.
- What does a computer do?
  - Fetch, Decode, Execute (FDX)
- Algorithm: step-by-step process to achieve desired result.
- Program: one or more algorithms to perform a single, specific task.
- Execute: running a program.
- Debug: fixing a 'bug' in your algorithm.

WHY PYTHON?

- **B**road, **E**asy, **E**fficient, **F**ast
- Interpreter: analyzes and executes code.
  - We tell Genie what we want, Genie gives us something back.
- Command shell/Console: environment for interacting with interpreter directly.
- >>> Prompt: where you type your command
- Python programs (or modules) are saved as .py files.
- Python 2.x vs. Python 3.x
  - **Not** backwards compatible: old software won't work with 3.x.
  - Syntax, libraries (collection of code), etc.
  - Not difficult to adapt to 3.x from 2.x.

GETTING STARTED

- Integrated Development Environment (IDE): "software application that provides comprehensive facilities to computer programmers for software development"
  - Ex. Writing an essay in Notepad vs. Word Document.
- IDLE
- Eclipse (PyDev)
- Spyder
- Notepad++
- Sublime
- Etc.
- Which one to use?
  - Whatever gets the job done.

MATH (IN YOUR CONSOLE)!

- 3 + (2 - 3) = 2
- (3 * 4) / 2 = 6
- 2 ** 3 = 8
- 64 % 10 = 4
- 64 / 10 = 6
- 64 / 10.0 = 6.4
- 64.0 / 10 = 6.4
- 64.0 / 10.0 = 6.4

- 64 // 10 = 6
- 64.0 // 10 = 6
- 4.5 * (6 / 10) =
  - **0.0**
- (3.0 * 4) - 13 * (3.0 / (4 / 5)) =
  - **ZeroDivisionError: float division by zero**
    - This is called a runtime error.

VARIABLES

- Placeholders for some value.
- `x = 2`
- `y = 10`
- `z = y ** x`
- `x = z ** (1 / x)`
- `What is x?`
  - **1**

- Naming conventions:
  - Descriptive
  - Cannot start with numbers, but can contain numbers: **`num1`**
  - No special symbols, except the underscore: **`first_name`**
  - **`firstName, lastName`**
  - No keywords
- Readability

WRITING A MODULE

- Python 2.x Shell
  - File → New File
- `x = 2 + 5`
- `x`
- Unlike the console, you need to **print** to see the **output**.
- `y = x + 5`
- `print x`
- `print y`

PRINT

- `x = 2`
- `y = 10`
- `z = y ** x`
- `print z`
- `x = z ** (1 / x)`
- `print x`

- `print “How many languages do you speak?”`
  - **How many languages do you speak?**
- `print ‘How many languages do you speak?’`
  - **How many languages do you speak?**
- `print “‘How many languages do you speak?’”`
  - **‘How many languages do you speak?’**
- `print ‘“How many languages do you speak?”’`
  - **“How many languages do you speak?”**

COMMENTS AND DEBUGGING TOOLS

- Comments help readability.
- Documentation is important!
- # This is a comment
- “““ This is a longer comment

  that goes to the next line. ”””
- Error: syntax, runtime, logic
- print is one of the most important debugging tools.
- IDE highlights syntax errors.
- Logic errors usually the hardest to fix.

TYPES

- Classification of the value that a variable can take.
- `TypeError: cannot concatenate 'str' and 'int' objects`
- Types include:
  - int
  - float
  - String
  - Boolean
  - List
  - Tuple
  - Dictionary
  - Etc.

- How to tell what type a value is?
  - type(*value*)
- What are functions?
  - `str(value)`
  - Pass an argument into parameter, typically the function returns something
  - `function_name(parameter)`
  - `help(function_name)`
- chr(*integer*)
- str(*value*)
- ord(*character*)
- float(*integer*)

STRINGS

- firstName = “Noam”
- lastName = “Chomsky”
- len(firstName) == len(“Noam”)
  - **True**
- len(lastName) != len(“Noam”)
  - **True**
- lastName = firstName
- len(firstName)
  - **4**
- len(lastName)
  - **7**

- first = “Nom”
- last = “Chompsky”
- name = first + last
  - name = “Nom” + “Chompsky”
  - **“NomChompsky”**
  - name = first + “ ” + last
  - **“Nom Chompsky”**
- first = first * 3
  - **“NomNomNom”**
- name = (first * 2) + last
  - **“NomNomNomNomNomNomChompsky”**
- **\n: newline (enter key)**
- **\t: tab (tab key)**

PUTTING IT TOGETHER!

- age = 85
- name = "Nomsky"
- sent = name + " is " + age + " years young."
  - **TypeError: cannot concatenate 'str' and 'int' objects**
    - What kind of error is this?
- str(1)
  - "1"
- Now, how can I get the sentence "Nomsky is 85 years young."?
  - **sent = name + " is " + str(age) + " years young."**
- sent = "My favorite color is " + str(len(name*2)+3) + "."
  - **"My favorite color is 15."**
- len(sent)
  - **24**

MANIPULATING STRINGS

- word = "bird"
- word[0]
  - 'b'
  - Why does the first element start at 0? Efficiency and arithmetic.
- word[2]
  - 'r'
- word[1]
  - **'i'**
- word[3]
  - **'d'**
- word[4]
  - **IndexError: string index out of range**

- word[-1]
  - **'d'**
- word[-4]
  - **'b'**
- len(word)
  - 4
- word[len(word)-1] ⇔ word[-1]
  - 'd'

0	1	2	3
B	I	R	D
-4	-3	-2	-1

MANIPULATING STRINGS (CONT.)

- sent = "I LIKE TURTLES."
- sent[1]
  - ' '
- sent[0:1]
  - 'I'
- sent[8]
  - 'U'
- sent[2:8]
  - 'LIKE T'
- sent[5:-1]
  - 'E TURTLES'

- sent[:6]
  - 'I LIKE'
- sent[-8:]
  - 'TURTLES.'
- sent[:]
  - 'I LIKE TURTLES.'
- sent[11:8]
  - ''
- How can I extract the word 'likes' from slicing sent?
- sent[0::2] = 'Ilk ute.'
- sent[1::2] = ' ietrls'

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I		**L**	**I**	**K**	**E**		**T**	**U**	**R**	**T**	**L**	**E**	**S**	**.**
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

STRING FUNCTIONS

- String functions typically in following configuration:
  - *string.function_name(parameter)*
  - What is the dot?
    - Object-oriented programming
- `word = "bird is the word"`
- `word.upper()`
- `word.lower()`
- `word.isalpha()`
- What is `word` now?
  - `word = word.upper()`

- *string*`.replace(`*old*`, `*new*`)`
  - `word.replace('i', 'a')`
    - **"bard as the word"**
  - `word.replace('bird', 'bard')`
    - **"bard is the word"**
- *string*`.find(`*substring*`)`
  - `word.find('d')`
    - `3`
  - `Word.find('q')`
    - `-1`
- *String*`.rfind(`*substring*`)`
  - `Word.rfind('d')`
    - `15`

STRING FUNCTIONS (CONT.)

- sent = " them dishes need washed "
- *string*.strip()
  - sent.strip()
    - **"them dishes need washed"**
- *string*.lstrip()
  - sent.lstrip()
    - **"them dishes need washed "**
- *string*.rstrip()
  - sent.rstrip()
    - **" them dishes need washed"**
- *string*.count(*substring*)
  - *sent.count('e')*
    - *5*
- *Python 2.7.8 Documenttion*
- *help())*

LISTS

- Collection of items, same data type or different data types.
  - What are data types? Integer, float, string, list, etc.
- Strings are similar to lists, but are not lists.
- `a = [1, 2, 3]`
- `b = ['a', 'b', 'c']`
- `c = [ ]`
- `d = [1, 'two', b]`
  - `[1, 'two', ['a', 'b', 'c']]`
- `len(c)`
  - `0`
- `len(d)`
  - `3`

- `x = a[0]`
  - `1`
- `x = x + a[2]`
  - `4`
- `d[0]`
  - `1`
- `d[1]`
  - `'two'`
- `d[2]` ⇔ `d[-1]`
  - `['a', 'b', 'c']`
- How to access the 'b'?
  - `d[2][1]` or `d[-1][1]`
- How to access the 'c'?
  - `d[2][2]` or `d[-1][2]` or `d[-1][-1]`

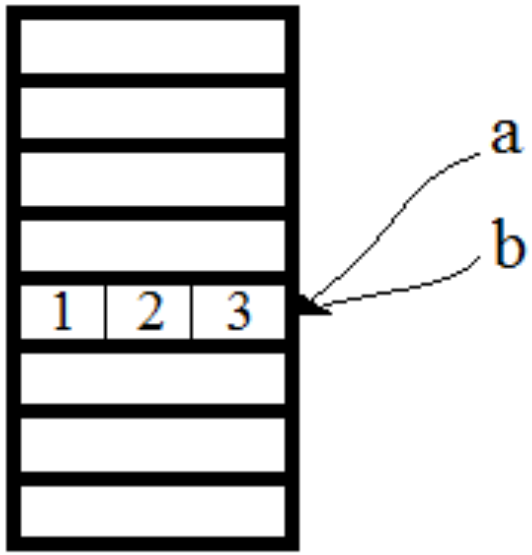
LISTS (CONT.)

- listA = ['a', 'b']
- listB = ['c', 'd']
- listA + listB
  - ['a', 'b', 'c', 'd']
- listA * 2
  - ['a', 'b', 'a', 'b']
- listA[1] = 'e'
- listA
  - ['a', 'e']
- listA = ['a', 'b', 'c', 'd']
- listA[1:3] = ['y', 'z']

- a
  - ['a', 'y', 'z', 'd']
- del a[1]
  - ['a', 'z', 'd']
- del a[1:3]
  - ['a']
- *list*.append(*item*)
  - a.append('b')
    - ['a', 'b']
- *list*.remove(*item*)
  - a.remove('a')
    - ['b']
- list.insert(0, 'a')
  - ['a','b']
- *Python 2.7.8 Documenttion*

ALIASING

- Lists are mutable, strings are immutable.
- If you want to alter a string, you will *need* to create a new string.
- x = “python is ”
- y = x
- y = y + “fun!”
- x
  - “python is ”
- y
  - “python is fun”
- a = [1, 2, 3]
- b = a
- b[1] = 1
- a
  - [1, 1, 3]



DOWNLOAD THE FOLLOWING TEXT FILE:

- http://eddersko.com/python/DecInd.txt

LOOPS

- `range(stop)`
  - **Or you can use** `xrange`
- `range(start, stop, step)`
- `range(5)`
  - `[0, 1, 2, 3, 4]`
- `range(2, 5)`
  - `[2, 3, 4]`
- `range(0, 10, 2)`
  - `[0, 2, 4, 6, 8]`
- **for** *var* **in** *list*:
  do something
- `for i in range(5):`
  `print i,`

- `a = [1, 2, 3, 4, 5, 6, 7, 8]`
- `for i in range(len(a)):`
  `a[i] += 2`
- `for x in a:`
  `print x + 2,`
- `sent = "ducks say moo"`
- `new = ""`
- `for x in sent:`
  `new += chr(ord(x)+1)`

NESTED LOOPS

- ```
  a = [ [1, 2], [3, 4], [5, 6], [7, 8], [9, 10] ]
  ```
- ```
  for x in a:
      for y in x:
          print y,
  ```
- ```
  for i in range(len(a)):
      x = a[i]
      for j in range(len(x)):
          print j[x],
  ```
- ```
  for x, y in a:
      print x, y,
  ```

- Best way to debug is to trace and print!
- Practice makes perfect (but perfection is impossible)!
- How do I print only the even numbers?
- How to add 5 to all the numbers?
- How to collapse into a single list?
- How to add numbers in each list and collapse into a single list?
  - i.e. [3, 7, 11, 15, 19]

INDENTATION

- Python cares about indentation (a lot).
-
```
for x in a:
for y in x:
   print y,
```

vs.

```
for x in a:
   for y in x:
      print y,
```

- I recommend using tabs all the time for indenting.
  - Mixing tabs and spaces is bad practice.
    - I learned that after 10 hours of debugging, and rewriting my whole program.

- Indentation helps readability.
- Make sure all your code has the same levels of indentation.
-
```
x = [1, 2, 3]
 str = "hello" ← wrong
a = ['a', 'b', 'c']
```

- IDLE (and most other IDEs) do a good job of spotting indentation errors, but keep an eye out for them yourself.

CONTROL FLOW (BOOLEANS)

- A loop is a type of control flow.
- Conditions (if statements) are another type of control flow, but first, Booleans!
- `2 < 5`
  - `True`
- `2 == 5`
  - `False`
- `2 != 5`
  - `True`
- `'python' == "PYTHON'`
  - `False`
- `'python' == 'py' + 'thon'`
  - `True`
- `['a', 'b'] != ['b', 'a']`
  - `False`
- `(2 > 5) and (2 < 5)`
  - `False`
- `(2 > 5) or (2 < 5)`
  - `True`
- `not(2 > 5) and (2 < 5)`
  - `True`

CONTROL FLOW (IF STATEMENTS)

- `a = [1, 2]`
- `b = [1, 2]`
- `a != b`
  - `False`
- not()
- What is the difference between `is not` and `!=` ?
- `1 in a`
  - `True`
- `3 not in a`
  - `True`

- `if (condition):`
  `do something`
  - In order to execute the code in the if-condition, the statement must be `True`
- `if True:`
  `do something`
  - When will this fail?

CONTROL FLOW (CONT.)

- `x = any number`
-
```
if (x == 5):
    x += 5

elif (x == 0):
    x += 10

else:
    x = (x ** 0)*10
```
- Whenever if-statements become unclear, use print statements to debug.
- `elif` and `else` are not always required.

- `letter = 'F'`
-
```
if score >= 90:
    letter = 'A'
```
-
```
if score >= 80:
    letter = 'B'
```
-
```
if score >= 70:
    letter = 'C'
```
- `print letter`
- Having said that, what's the problem here?

PUTTING IT TOGETHER!

- **Some nifty functions…**
- `input("Enter: ") → value`
  - `You typed in 4.`
    - `Computer reads 4 (as an integer).`
- `raw_input("Enter: ") → string`
  - `You typed in 4.`
    - `Computer reads "4" (as a string).`

INPUT/OUTPUT

- Also known as I/O.
- Communication between you and your computer.
- `input` and `raw_input` are inputs.
  - Keyboard and mouse.
- `print` is a type of output.
  - Monitors and printers.
- Essentially, we input, the computer processes (and stores), then outputs.
- Another type of I/O is File I/O where communications happen between the program and some files.
- File processing is important and useful!

FILE PROCESSING

- `a = raw_input("Enter: ")`
- `b = input("Enter: ")`
- What is the difference?
  - Use `type(value)`
- Reading a file is like reading a book.
  - 1. Open the book.
  - 2. Read the first line.
  - 3. Read the second line.
  - 4. Read the third line.
  - 5. …
  - 6. Read the last line.
  - 7. Close book.

```
# open the file to 'r'ead

# file needs to be in the same
folder

infile = open('file_name.txt' , 'r')


# loop through all the lines in the
file

for line in infile:

    # print line and name

    print line

infile.close()
```

FILE PROCESSING (CONT.)

- When you press [**Enter**], you are actually typing: `\n`
  - [**Tab**] : `\t`
- `sentence = "My name is Edwin."`
- `sentence.split(" ")`
  - ["My", "name", "is", "Edwin."]
- `sentence = "My  name"`
- `sentence.split("\t")`
  - ["My", "name"]

- Writing a file
- `outfile = open(`*`filename`*`, 'w')`
- `print >> outfile,` *`content`*
- `outfile.close()`
- Let's Practice!

PUTTING IT TOGETHER!

- How many lines are in the file?
- Print all the particles.
- How many 'up' particles are there?
- Print all the distances between the verb and particle.
- What is the average distance between the verb and particle?