UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

3250 Foundations of Data Science

Module 2: Introduction to Python

Course Plan

Module Titles
Module 1 – Introduction to Data Science
Current Focus: Module 2 – Introduction to Python
Module 3 – NumPy
Module 4 – Pandas
Module 5 – Data Collection and Cleaning
Module 6 – Descriptive Statistics and Visualization
Module 7 – Workshop (No Content)
Module 8 – Time Series
Module 9 – Introduction to Regression and Classification
Module 10 – Databases and SQL
Module 11 – Data Privacy and Security
Module 12 – Term Project Presentations (no content)

Learning Outcomes for this Module

- Explore the history of Python
- Understand its basic syntax
- Use Python to write simple programs to manipulate data

Topics for this Module

- **2.1** Introduction to Python
- **2.2** Installing Python
- **2.3** Jupyter
- **2.4** Variables and expressions
- **2.5** Functions
- **2.6** Conditionals
- **2.7** Iteration
- **2.8** Strings
- **2.9** Containers
- **2.10** Resources
- **2.11** Homework

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 1

Introduction to Python

Python for Data Science

- #1 for social media hacking
- Python is a general purpose language so more versatile than R
- The **pandas** library provides capabilities similar to those of the R data frame
- **SciPy** and **Scikit-learn** libraries (& many others) provide stats and machine learning algorithms

Python for Data Science (cont'd)

- Nice libraries for parsing natural language text (**NLTK** – Natural Language Toolkit), web page contents, tweets, etc.
- Can link to subroutines in C or that run on a GPU where performance is critical
- Overview of common Python myths with examples, from Mahmoud Hashemi, a lead developer of the Python Infrastructure team at eBay/PayPal: 10 Myths of Python

History of Python

- Initially conceived of and created by Guido van Rossum in the late 1980's
- Named after Monty Python's Flying Circus
- Interpreted and dynamically typed
- Supports a mixture of procedural, object oriented, functional and imperative styles
- Has a reputation of being relatively easy to learn
- Quite elegant: consistent syntax
- Latest stable release is 3.6 but 2.7 is in wide use

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 2

Installing Python

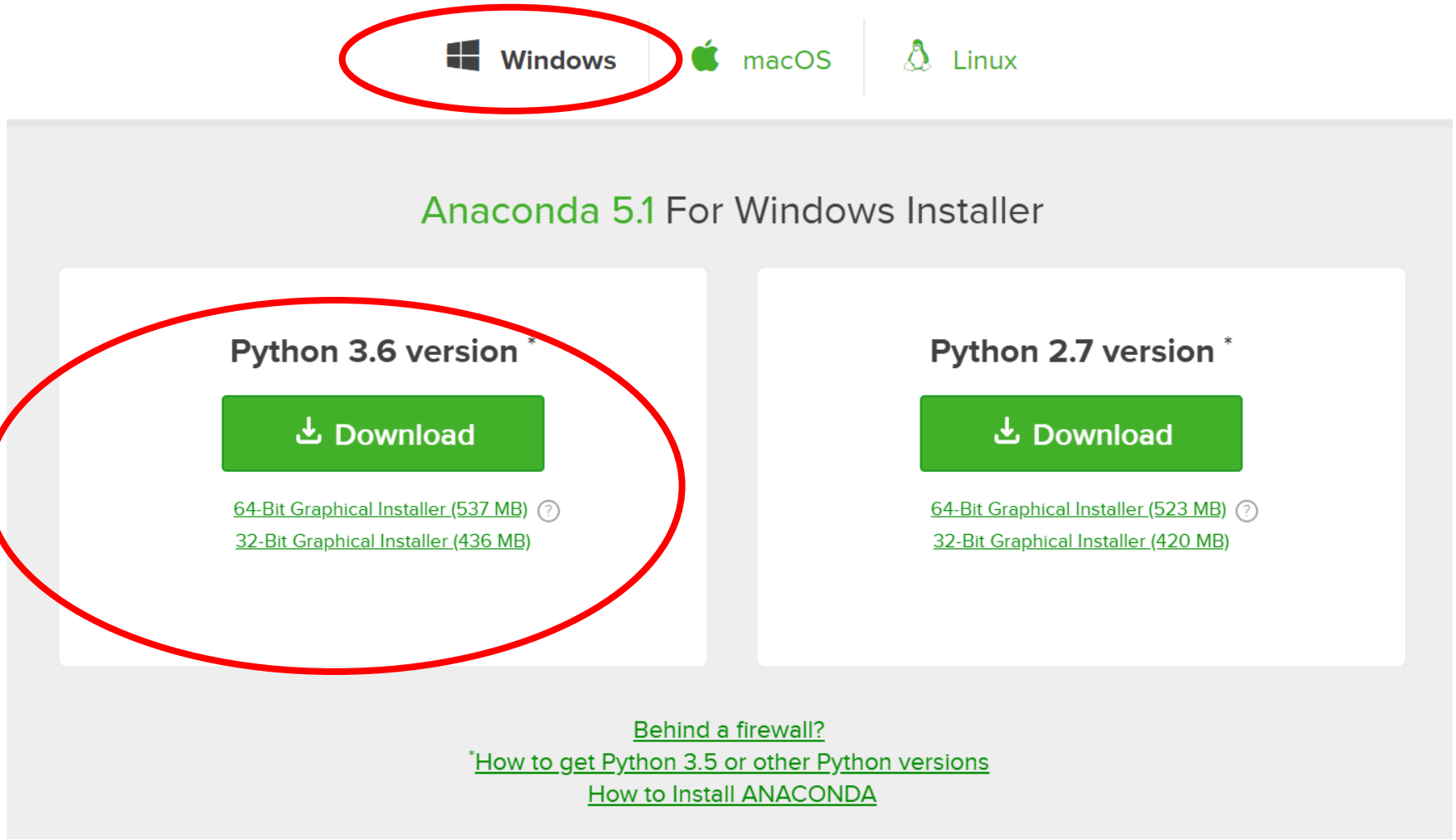
The Anaconda Distribution

- Anaconda is a free distribution of Python for scientific/statistical computing curated by Continuum Analytics
- Provides isolation so won't interfere with any other Python installation you already have
- Allows you to install multiple versions of Python on the same machine if needed

Anaconda

- If you haven't already, install Anaconda
- It can be downloaded from www.continuum.io
- Install the Python 3.6 version
- Anaconda is a free distribution

Download Anaconda for Windows

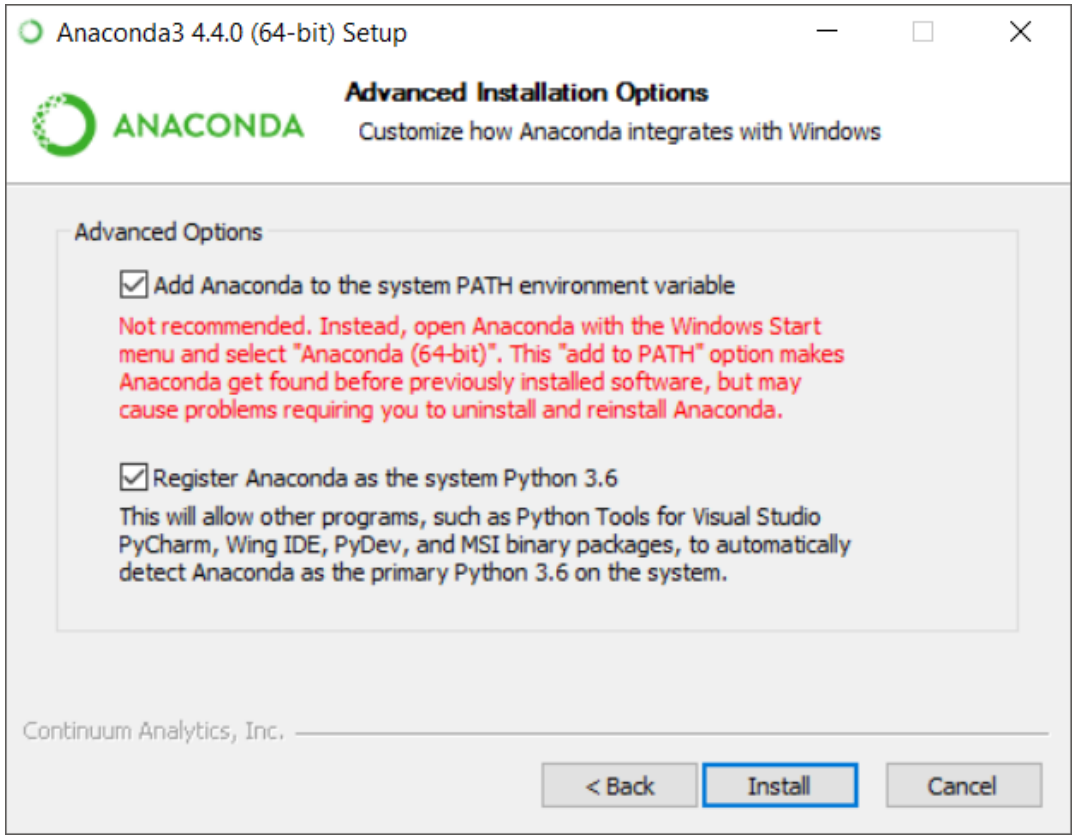


Source

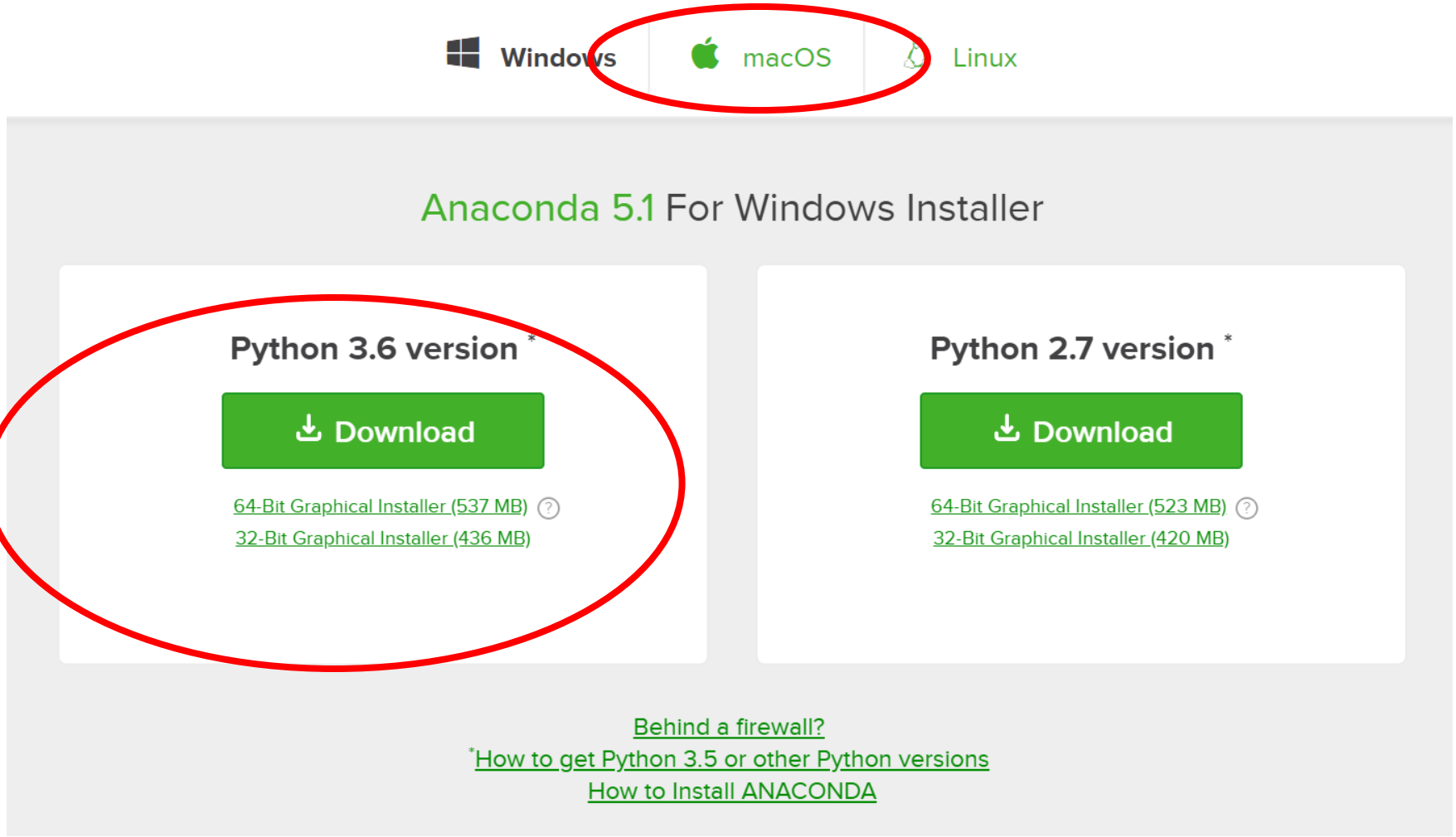
Windows Installation

- Make sure to **select** the first checkbox on Advanced Installation Options screen, “Add Anaconda to the system PATH environment variable”

- Install Anaconda into a path that does not contain spaces or unicode characters.



Download Anaconda for Mac



Source

Installing Anaconda

- After you download the installer, start the installation and follow the prompts to install to the default location
- If you encounter any issues, consult documentation - (Windows) or (Mac)

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 3

Jupyter

- A “notebook”
- Stands for Julia + Python + R
- Formerly “IPython notebook”
- Alternatives:
  - Beaker
  - Anaconda Spyder
  - IDE’s e.g. PyCharm

Launching Jupyter

- Open a command window (Windows) or Terminal (Mac/Linux)
- Change directory to Documents or the working directory for this course
- Type `ipython notebook` or `jupyter notebook` (whichever works) to launch Jupyter

Jupyter Hints

- Put all your notebooks together in a folder for this course
- Up/down keys to jump between cells
- Shift-enter to execute a cell
- If the last line in a cell returns a value it will print
- Mark cells as code or text
- If the cell has a * for the number and the little 0 in the top right is filled in Python is busy

More Jupyter Hints

- `%quickref` for shortcuts
- Use `%matplotlib inline` to have plots display in the notebook rather than in a separate window
- Use shift-tab to see the signature (calling sequence) of a function

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 4

Variables and Expressions

Variables, Types and Values

- Variable names
  - Must start with a letter
  - Typically all lower case with _ between words (“snake case”)
  - Are case sensitive
  - Python 3 has 35 reserved keywords
- Main types used in data science
  - Boolean
  - Integer
  - String
  - Float

Determining Types

- It is often useful to verify the type of a variable
- Checking type of the variable: `type()`
- Checking if a variable is of certain type can be achieved by using the `isinstance()` Boolean function
- Example

```
x = 1
isinstance(x, int)
```

Expressions and Statements

- Python supports the following operators on numbers:
  - **+** addition
  - **-** subtraction
  - * multiplication
  - / division
  - ** exponent
  - % remainder, or modulo
- Usual order of operations:
  - Parentheses ()
  - Exponentiation **
  - Multiplication/Division
  - Addition/Subtraction
- Expression has a value; statement doesn't

Module 2 – Section 5

Functions

Functions

- Can be true functions or subroutines
- Defined using `def(parm1, parm2, …):`
- Returns value after `return`
- Example

```
def f(a, b):
    a = a + 2
    return a ** b
```

- Note the indentation
- Once a return line is executed, the function will terminate and return to where it was called from

Declaring & Using Functions

- Declaring:

```
def f(a, b):
    if (a + b) > 10:
        return True
    else:
        return False
```

- Invoking (using):

```
f(4, 7)
```

Module 2 – Section 6

Conditionals

Conditionals

- Boolean Expressions
  - Can only be `True` or `False`
  - The == operator tests for equality
  - Not equivalent to a string with the same value

- Example

```
a = “apple”
b = “orange”
a == b
```

Conditionals (cont'd)

- Logical Operators
  - Three exist in Python: `and`, `or`, `not`
  - Similar meanings to English
  - Returns a Boolean expression

- Example

```
x = True
y = False
x or y (return true if either x or y is true)
```

Conditionals (cont'd)

- Conditional Execution
  - Allow the execution of code based on certain conditions
  - Most common conditional statements: `if`, `else`
  - `if` statements will execute associated code if its conditions are true at the time of evaluation otherwise the else code will be executed

- Example

```
score = 92
if (score > 50):
    print("You passed!")
else:
    print("You failed.")
```

Conditionals (cont'd)

- Chained Conditionals
  - `elif` (else if) statements may be used to create additional conditions which will be evaluated in order
  - There is no limit to the number of elif statements

- Example

```
a = 10
if (a > 10):
    print("a is greater than 10")
elif (a < 10):
    print("a is less than 10")
else:
    print("a is equal to 10")
```

Conditionals (cont'd)

- Nested Conditionals
  - Within a block of code associated with an `if…else` statement can be additional `if…else` statements creating more complex code execution logic

- Example

```
age = 31
if (age >= 30):
    if (age < 40):
        print(”He is in his 30s”)
    else:
        print(“He is not in his 30s”)
else:
    print(“He is not in his 30s”)
```

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 7

Iteration

Iteration

- Variables may be assigned values multiple times with the variable's value updating each time to reflect its new value
- Example

```
x = 5
print(x)
x = 10
print(x)
```

Iteration (cont'd)

- The `while` statement allows for efficient repetition of code. It first evaluates whether the given condition:
  - If true: executes associated code and returns to revaluate condition
  - If false: ends the loop and continues to the next statement in the program
- Example

```
def counter():
    n = 0
    while (n < 100):
        print("n is equal to ", n)
        n += 1
```

Iteration (cont'd)

- Infinite Loops
  - Occur when a loop begins, but is written so that the condition can never evaluate to false and thus continues endlessly
- Example

```
def counter:
    n = 0
    while (n < 1):
        print("n is equal to " + n)
```

- Without incrementing the value of n at the end of the loop, the `while` statement will always evaluate to true

Iteration (cont'd)

- The `break` statement allows for a loop to be exited prematurely
- This statement can be useful when the programmer doesn't know the number of iterations the loop must make
- Example

```
def counter(n):
    while (n < 1000):
        if ((n * 11) % 7 == 0):
            print (n)
            break
        else:
            n += 1

counter(15)
```

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 8

Strings

Strings

- Strings are a sequence of characters
- Individual characters in a string can be accessed using the letter's index with the first character having an index value of 0
- Example

```
name = "Michael"
name[0] ("M")
name[3] ("h")
```

- The len function will return the number of characters in a string

```
name = "Michael"
len(name) (7)
```

Strings (cont'd)

- Transversal
  - The process of accessing all the items in a set exactly once each
  - The `for` loop allows for compact transversal structures
  - The example `for` loop selects each character in name then stores it in the `char` variable and prints the character
- Example

```
name = "Michael"
for char in name:
    print(char)
```

Strings (cont'd)

- String Slices
  - A portion of a string selected with the operator `[x:y]` where x is the index of the start of the slice and y is the index of the end of the slice
  - x is inclusive and y is exclusive and therefore the yth element is not included in the string slice
  - Omitting x defaults to 0 and omitting y defaults to the length of the string
- Example

```
name = "Michael"
name[2:5] ("cha")
name[:3] ("Mic")
name[1:] ("ichael")
```

Strings (cont'd)

- Searching
  - A type of traversal where we attempt to locate something and returning where it is found
- Example

```
def search(num, long_num):
    index = 0
    while (index < len(long_num)):
        if (long_num[index] == num):
            return index
        index += 1
    return -1
```

Strings (cont'd)

- Counters
  - Counts occurrences of something by incrementing a count variable whenever the occurrence is found
- Example

```
name = "Mississauga"
count = 0
for char in name:
    if char == "s":
        count += 1
print(count)
```

Strings (cont'd)

- String methods
  - A method is similar to a function, but uses different syntax
  - There are many methods that can be invoked on strings using the syntax `string_name.method_name(params)`
- Example

```
food = "PIE"
print(food.lower())
```

Strings (cont'd)

- The `in` operator
  - A Boolean operator that return true if the first of two strings is a substring of the second
- Example

```
"a" in "apple"
```

Strings (cont'd)

- String comparison
  - Can be done with == operator for equality or with >, <=, etc.
  - The inequality operators check for alphabetical order with all uppercase letters coming before any lowercase letters in the ordering
- Example

```
my_letter = "C"
my_letter == "C" (true)
A > my_letter (true)
a > my_letter (false)
```

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 9

Containers

Lists

- Lists are a sequence of values similar to a string
- A string is a sequence of characters, but a list is a sequence of any type
- Lists can even contain other lists as an element thereby nesting lists
- Unlike strings, lists are mutable and elements can be changed
- Example

```
my_list = [0, 2, 4]
new_list = [my_list, [0, 1]]
```

Lists (cont'd)

- One can transverse lists, operate upon them using `+` and `*`, slice them, and invoke numerous methods upon them
- The `pop` method deletes an element at the given index and the list method converts a given string into a list
- Example

```
my_list = [0, 2, 4]
new_list = 2 * my_list ([0, 2, 4, 0, 2, 4])
my_list[0:2] ([0, 2])
my_list.count(0) (1)
my_list.pop(2)
my_list [0, 2]
```

Dictionaries

- A dictionary is similar to a list where the indices are not limited to only integers
- The dictionary is a set of key-value pairs where the key is the index to its associated value
- Example

```
my_dictonary = {“a” : “apple”, “b” : “banana”}
print(my_dictonary[‘a’]) → apple
```

Tuples

- Tuples are similar to lists, but are immutable
- A tuple is declared as a comma-separated sequence of values or using the tuple function
- Example

```
new_tuple = “apple”, “banana”, “carrot”
t = tuple(“trouble”)
print(t) →(“t”, “r”, “o”, “u”, “b”, “l”, “e”)
```

Tuples (cont'd)

- Tuple Assignment
  - Swapping the values of variables is simple using a tuple assignment
- Tuple Return Values
  - Returning a tuple can allow for a function to return more than one calculation's result even though the function can only return one value
- Example

```
a = "art"
b = "bass"
(a, b) = (b, a)
print(a) → ("bass")
print(b) → ("art")
```

Questions?

Any questions?

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 2 – Section 10

Resources

Resources: Introductory

- Python home page
- Python vs. R: Python vs R by packtpub.com
  Python vs R for Data Science
- Python Cheat Sheets:
  - Google Drive doc
  - Blog by Sebastian Raschka
  - Cheat Sheet by Kailash Ahirwar

Resources: Introductory (cont'd)

- Ceder, Naomi. The Quick Python Book, Second Edition: Covers Python 3. Manning Publications.

- Quick reference sheets for working in IPython and Jupyter
- Think Python

Resources: Introductory (cont'd)

- PyData, a forum providing a venue for users across all the domains of data analysis to share ideas and learn from one another: pydata.org/
- Anaconda download: www.continuum.io/downloads
- Gries, Campbell & Montojo. Practical Programming (2nd edition) - An Introduction to Computer Science Using Python 3: pragprog.com
- Coursera. Learn to Program: The Fundamentals coursera.org

Resources: Introductory (cont'd)

- Learn Python the Hard Way: learnpythonthehardway.org/
- PEP 8 -- Style Guide for Python Code - coding conventions for the Python code: www.python.org/dev/peps/pep-0008/
- Automate the Boring Stuff with Python: automatetheboringstuff.com/

Resources: Advanced

- Python Cookbook (version 3):
- Problem Solving with Algorithms and Data Structures:
- Debugging in Python - steps you can take to try to debug your program:

Module 2 – Section 11

Homework

Python Review

- Review the Python Intro Notebook we worked on in class, Think Python and, for more examples: Python Intro Notebook

- Do not submit the notebooks for this part of the homework

Assignment 1

Part 1. Loops and Lists

1.1. Write a for-loop which will read a list of any kind and print out each item in a list on a separate line.

For example, if you start with the list of fruits:

fruits = ['apples', 'oranges', 'pears', 'apricots'],

the output of the code can be:

a fruit of type: apples

a fruit of type: oranges …. etc.

The code should work with a list of any length and any type of list, i.e. strings, integers, other lists. Test your code with at least 2 different types of lists.

1.2. Start with an empty list and generate a list of integers from 0 to 5, reverse the list and print it out. Explain how you reversed the list.

Assignment 1 (Cont'd)

Part 2. Dictionaries

- Create two dictionaries, ***provinces*** and ***capitals***.
  - Dictionary ***provinces*** will use province names as a key and province abbreviation as a value.
  - Dictionary ***capitals*** will use province abbreviation as a key and a capital name as a value.
- Print out all combinations of the province names and the corresponding capitals.

  An example of the output:

  The capital of Alberta is Edmonton

  The capital of British Columbia is Victoria
- Add 2-3 provinces and corresponding capitals to your dictionaries. Print out the resulting list of province/capital combinations.

Assignment 1 (Cont'd)

Part 3. Greatest Common Divisor (GCD)

- The Greatest Common Divisor (GCD) of *a* and *b* is the largest number that divides both of them with no remainder. You can read about GCD here
- Write a function called ***my_gcd()*** that takes parameters *a* and *b* and returns their greatest common divisor.
- Create your own implementation of the algorithm and test your code by comparing the output of your function to Python's ***gcd()*** function from ***math*** package.
- Test the code with the following inputs: (39, 91), (20, 30) and (40,40). Explain the logic of your function.
- *Hint:* You might want to review Think Python book, chapters 5 and 6, sections that talk about recursive functions.

Assignment 1, Medium Complexity – Birthday Paradox

- Birthday paradox concerns the probability that, in a set of *n* randomly chosen people, some pair of them will have the same birthday. You can read about this problem here
- Write a function called *`has_duplicates`* that takes a list and returns *True* if there is any element that appears more than once. It should not modify the original list.
- If there are 45 students in the class, what are the chances that any two students will have the same birthday? You can estimate this probability by generating random samples of 45 birthdays and checking for matches.

Assignment 1, Medium Complexity – Birthday Paradox (Cont'd)

- *Hints*: (1) you can generate random birthdays with the `randint` function in the `random` module. (2) For simplicity, use the day number of the year, not the actual date, (3) you can use the book solution as a starting point for this assignment:
- **The code should print out**:
  - number of students,
  - number of iterations/samples
  - list of duplicate days for each iteration, where duplicates are found.

Submitting Assignments

- Submit only one assignment, either Assignment 1 (all 3 parts), or the Birthday Paradox
- Submit the Jupyter notebook with your code to Blackboard
- There are no extra points awarded if you submit more than one assignment
- The assignment is due at the beginning of Class 4
- Late assignments will count for half the marks

Next Class

- Introduction to NumPy
  - The Numerical Python (NumPy) library
  - Indexing and Slicing a NumPy array

Follow us on social

Join the conversation with us online:

facebook.com/uoftscs

@uoftscs

linkedin.com/company/university-of-toronto-school-of-continuing-studies

@uoftscs

Any questions?

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Thank You

Thank you for choosing the University of Toronto School of Continuing Studies