SOFTWARE: COMPUTER PROGRAMMING BASICS II

SOOK-LEI LIEW, PHD, OTR/L, SLIEW@USC.EDU

REVIEW OF LAST WEEK

- Last week, we:
  - Reviewed the course outline and objectives
  - Installed Anaconda
  - Learned about Jupyter
  - Wrote our first Python commands
  - Wrote simple single-line commands to print to screen and to perform basic operations
  - Briefly reviewed a Python IDE (Spyder)

OBJECTIVES THIS WEEK

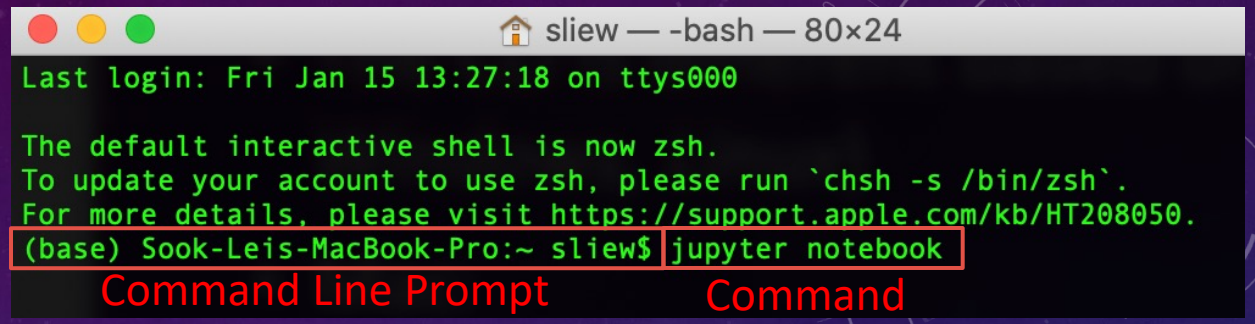
By the end of this session, students will understand and implement:

- Command Line Prompts
- Basic data types
- Variables
- Conditionals
- Syntax
- Conditional statements:
  - If/Then statements
  - For loops
- Basic functions and libraries
- Reading/writing CSV files (input/output)

COMMAND LINE PROMPTS

- Anaconda Navigator is a GUI (graphical user interface)
- We can open Anaconda Navigator > Jupyter Notebook
- But, we can also use a CLI (command line interface) to open Jupyter Notebook
  - Typing a line of code will have the same effect as pressing a button/clicking an icon

COMMAND LINE PROMPTS



- The CLI may look/be different based on your operating system (*Mac, Windows, Linux*)
- Mac:
  - Click the search icon at the top right corner
  - Type in “Terminal”
  - Open the “Terminal” application
  - Type in “jupyter notebook” at the prompt (all lowercase)

COMMAND LINE PROMPTS

- Windows:
  - Click the start/search button at the bottom right corner
  - Type in “Anaconda Prompt”
  - Open the “Anaconda Prompt” application
  - Type in “jupyter notebook” (all lowercase)

COMMAND LINE PROMPTS

- Linux:
  - Open the native terminal
  - Type in “jupyter notebook” (all lowercase)

COMMAND LINE PROMPTS & YOUR OS

- Mac has a native Linux-like system built in plus a more friendly GUI-based system on top of it
- Windows does not really have native terminal access
- Another difference is file paths (/ on mac vs \ on windows)
- Therefore, in this class, since there are a mix of OS, we will use language-specific IDEs (Python → Jupyter Notebook, Spyder) so we don't have to show everything on Windows vs Mac

BASIC DATA TYPES

- In addition to binary to understand what you're typing, computers also need to know what **data type** the input you provided is
- Each data type has different properties, rules, and operations
- Pause here to watch this short Youtube video (also included in the Youtube description):
- https://www.youtube.com/watch?v=A37-3lflh8I

BASIC DATA TYPES

- Different data types have different properties and are allowed for different operations
- For instance, numbers (type: integer, float) can be used with math operations, but strings can't because it wouldn't make sense
  - 5/5 = 1
  - "hello"/"goodybye" = ??? ERROR
  - Note – typo in video "11" + "1" = "111"
  - Note: " " denotes string
    - "1" is string, 1 is integer

BASIC DATA TYPES

- Common data types include:
  - Integer (-1, 0, 1, 2, 3)
  - Float (-1.99, 0.01, 1.01)
  - String (a, b, c, d, words)
  - Boolean (True, False)

VARIABLES

- However, you can assign a value to a string:
  - X=5
  - Y=4
  - X+Y = 9
- In this case, X and Y are called **“variables”** because they represent an assigned value. The benefit of this is that the assigned value can change easily.
- You can assign a variable to be an integer, float, string, or Boolean, etc.
- Another example:
  - X=“Hello”, Y=“World”, X+” “+Y = “Hello World”

CONDITIONALS

Sometimes you want the computer to do something **only** when a variable meets a certain condition

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

SYNTAX

- One of the most common issues that has to be debugged is **incorrect syntax**
- Similar to sentence structures in English, the computer needs commands in a specific order and format
- Commands should be typed with precision, noting capitalization, semicolons, operations, etc. (remember binary!) – otherwise: error
- In Python, spaces and tabs matter! Tab tells the computer you aren’t done with your command

CONDITIONAL STATEMENTS

- Often, you will want the computer to do something when a variable meets a certain requirement.
- Two common ways to write this are:
  - If/then statements (if x is true, then do y)
  - For loops (while x is true, then do y)

IF STATEMENTS

```
a = 33
b = 200
if b > a:
    print(“b is greater than a”)
```

IF/ELSE (ALSO CALLED IF/THEN) STATEMENTS

Else If → elif

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

IF/ELSE (ALSO CALLED IF/THEN) STATEMENTS

Else

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

IF/ELSE (ALSO CALLED IF/THEN) STATEMENTS

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

IF/ELSE REVIEW

- if
- if / elif
- if / elif / else
- if / else

FOR/WHILE LOOPS

- As long as a conditional is true, do X
- Considered “iterative” as it will iterate (or repeat) command(s) for as many inputs as it is given
- Great for repetitive tasks
- Basic syntax:
- for x in ”banana”:
  print(x)

WHILE LOOPS

Can also be used to count through items

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

BASIC FUNCTIONS

- Usually, we will want to do more complicated commands, which require many commands
- Instead of writing the same commands over and over, these can be written as a **function**
- **Functions** can be called, and will do all the commands in the function then return to the code
- **Functions** can take inputs and can provide outputs (but don’t have to)

BASIC FUNCTIONS

The inputs to a function are called parameters or arguments and can vary

print(x)

print(“hello”, “world!”, sep=“ ”)

Syntax

```
print(object(s), sep=separator, end=end, file=file, flush=flush)
```

Parameter Values

Parameter	Description
object(s)	Any object, and as many as you like. Will be converted to string before printed
sep='*separator*'	Optional. Specify how to separate the objects, if there is more than one. Default is ' '
end='*end*'	Optional. Specify what to print at the end. Default is '\n' (line feed)
file	Optional. An object with a write method. Default is sys.stdout
flush	Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

EXAMPLE: THE RANGE() FUNCTION

Definition and Usage

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Syntax

```
range(start, stop, step)
```

Parameter Values

Parameter	Description
start	Optional. An integer number specifying at which position to start. Default is 0
stop	Required. An integer number specifying at which position to stop (not included).
step	Optional. An integer number specifying the incrementation. Default is 1

EXAMPLE: THE RANGE() FUNCTION IN A FOR LOOP

Python range() Function

‹ Built-in Functions

Example

Create a sequence of numbers from 0 to 5, and print each item in the sequence:

```
x = range(6)
for n in x:
  print(n)
```

- Note: Python is a 0-indexed language, meaning it counts the first value as 0 (not 1)
  - range(6) → 0, 1, 2, 3, 4, 5
  - range(2,6) → 2, 3, 4, 5 (2 is starting, 6 is ending)
  - range(2,6,2) → 2,4 (2 is starting, 6 is ending, and it takes steps of 2)

MODULES, PACKAGES, AND LIBRARIES

- Functions usually do one simple task
- **Libraries** are bundles of code that may have hundreds of functions and do lots of things, usually around a theme, such as:
  - os: a module that contains operating system interfaces
  - csv: a module that supports reading and writing csv files
  - pandas: a library for data analysis and manipulation, including reading and writing csv files
  - matplotlib: a library for plotting data
  - numpy: a library for more complex mathematical operations and statistics

MODULES, PACKAGES, AND LIBRARIES

- When we work with modules/packages/libraries, we need to import them (otherwise, python would be massive) using the import command

```
import os
import pandas as pd
```

INSTALLING PACKAGES

- Some packages/libraries are “native” to python, or so regularly used that they are included when you download python.
- Others need to be installed separately (otherwise the download would be massive)
- Generally to download a new package with anaconda, you can type:

conda install [package name]

conda install pandas

MODULES, PACKAGES, AND LIBRARIES

- Then, when we call a function from a library, we reference the function

os.getcwd()

pd.read_csv(filename)

mydf.to_csv(filename) (<-- updated/fixed after lecture)

INPUTS & OUTPUTS

- Input: Computers can take in user input from the command line, or they can read input from files
- Files with lots of formatting that make it easier for humans to read (e.g., Microsoft word) have a LOT of extra code in them that make it harder for computers to read
- Some of the more common computer-readable files are .txt (text) files and .csv (comma-separated values) files

READING AND WRITING CSV FILES

- A computer can **read in** data from a CSV file
- Then we can manipulate that data (e.g., as a data frame) as integers, strings, etc.
- The computer can then **write out** the data frame back to a CSV file
- Other inputs: matricies (.mat), text files (.txt), json files (.json), etc.

READING AND WRITING CSV FILES

In python, the pandas library is one useful option (but there are many!):

(may need to first do: conda install pandas)

```
import pandas as pd
mydf = pd.read_csv("week2.csv")
mydf.head(5)
```

[can do additional operations to manipulate the file]

```
mydf.to_csv("week2_updated.csv", index=False)
```

WRITING YOUR OWN FUNCTION

```
def my_function(x):
    y=x+10
    print(y)
    return y

a=my_function(1)
print(a)
```

definition function_name(input parameters)

what you want the function to do with input

what you want the function to do with input

what you want it to return (optional)

OTHER PYTHON IDES

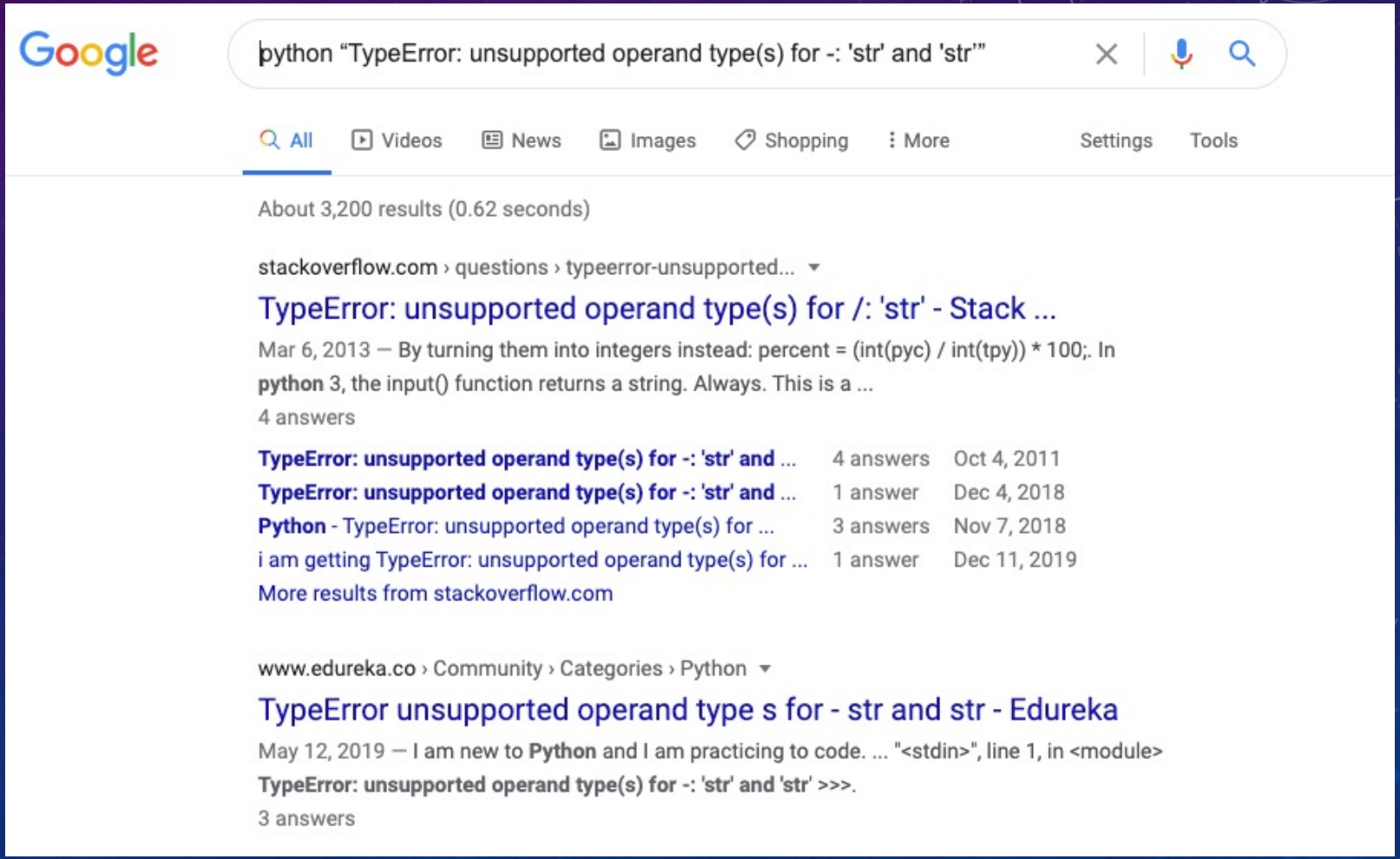
- Here we used jupyter notebook because it is a quick and easy way for you to get started with short commands and seeing output quickly
- For more involved programming (e.g., many lines of code), you would use something more like Spyder (also in Anaconda Navigator)
- We briefly looked at Spyder in the lab, as you may use it later in the class → you will see your variables, can debug, etc.

ASSIGNMENT AND TROUBLESHOOTING

- One of the most important parts of programming is learning to figure out errors and think through logic on your own!
- Here we gave you most of the code, but you can play around, try different inputs, tweak the commands slightly, etc.
- **Try the assignment to write your own function** to will output whether the number you give your function is greater than equal to 10 or not
- **Write the function and three test cases** (how would you know if it works correctly or not)

TROUBLESHOOTING

- If you get an error, or don’t know how to do something, Google is your best friend ☺
- Google the language + the error in quotes



TROUBLESHOOTING

- StackOverflow is your friend!
- Googling questions also helps you to learn the right way to ask questions

python pandas how to multiply a column value

All | Videos | Shopping | News | Images | More | Settings | Tools

About 802,000 results (0.65 seconds)

Use the * operator to multiply a column by a constant number

Select a **column** of DataFrame df using syntax df["column_name"] and set it equal to n * df["column_name"] where n is the number to **multiply** by.

www.kite.com › python › answers › how-to-multiply-a-c...

How to multiply a column in a pandas DataFrame by a scalar ...

About featured snippets | Feedback

stackoverflow.com › questions › python-pandas-datafra...

Python: Pandas Dataframe how to multiply entire column with ...

Sep 27, 2017 — copy() . The problem is already stated **in** the error message you got " SettingWithCopyWarning: A **value** is trying to be set on a copy of a slice from ...

11 answers

Multiplying columns with a constant **value in pandas** ...	1 answer	Mar 1, 2017
Pandas Multiply Specific **Columns** by **Value In Row** ...	2 answers	Oct 11, 2018
how to **multiply** multiple **columns** by a **column in** ...	2 answers	Aug 29, 2020
I want to **multiply** two **columns in** a **pandas DataFrame** ...	7 answers	Dec 28, 2012

More results from stackoverflow.com

GOOD LUCK!

- Programming is SO MUCH trial and error, and googling every thing you want to do
- Once you do it for a long time, you start to remember specific commands, and learn how to find the information that you need
- I still google most things that I want to do, unless I do them ALL the time ☺