A Guide to Python Concepts and Notebooks
PHIL 3050: Information and Uncertainty
Don Fallis

This handout provides a guide to the Python notebooks used in this class with summaries of the major concepts. For further details and code examples please refer to the notebooks.

Notebook Outline

Below is an outline of notebooks used in this class with major concepts in bold. This outline can be used as a quick reference or to help you find what you are looking for in the notebooks.

Notebook 1: FA24_Notebook_1_PythonFundamentals.ipynb

- **Colab notebooks**: Colab notebooks contain code and text cells (like paragraphs in an essay) that let you write and run code, as well as add explanatory text.
- **Expressions and operators**: Expressions in Python combine values using operators to create a single value. For example, `1+3` is an expression that combines the values `1` and `3` using the `+` operator to create the value `4`. You can also use other operators such as `*` to multiply numbers or `/` to divide numbers.
- **Strings**: A segment of text in Python is called a string. A string can be written with either single or double quotes, as long as it begins and ends with matching quotes, and they need to be the "straight" version, not curly/smart quotes. For example, `'1700 butterflies!'` and `"Hello world"` are strings.
- **Integers**: An integer is a whole number without a decimal point. For example, `1700` is an integer.
- **Floats**: A float is a number which uses a decimal. For example, `1700.5` and `1700.0` are floats.
- **Variables**: A variable is like a container that stores information. For example, when we write the assignment statement `bowl = 'soup'` we are using the variable `bowl` to store the string `'soup'`. We can also use variables to store other data types. For example, `cats_in_house = 5` stores the integer `5` as the number of cats in the house. The assignment statement `menu = {'cookie': 2,'croissant': 5}` stores the prices of items as a dictionary in the variable `menu`. In Python, you don't have to tell the computer what type of data the variable is holding since the compiler will determine that by itself.
- **Functions**: A function is a collection of code which can be applied and reapplied to complete repetitive or similar tasks. For example, `print()` is a function that prints out whatever is inside the parentheses when it is run. You can also write your own

functions (see our functions handout). The below code provides an example of a function to add an ingredient to a soup:

```
def recipe(ingredient):
    cooking = 'vegetable_soup'
    soup = ingredient + cooking
    return soup
```

Notebook 2: FA24_Notebook_2_Functions&Iteration.ipynb

- **Libraries and Modules**: Libraries and modules contain functions that other people have written. When we import libraries and modules we can use these functions.
- **Writing functions**: We can write our own functions by starting with the header line to name the function and specify any inputs, then writing statements defining what we would like the function to do, and concluding with a return statement to define the function output. Notebook 2 also contains a general review of functions. See the Python functions handout for more information.
- **Parameters and arguments**: Parameters are placeholders for values that are input into a function, and are used in function definitions; arguments are the actual input values.
- **Iteration with `for` loops**: `For` loops are one way to perform iteration in Python. A `for` loop is a series of instructions, or things to do, that you want to perform on multiple elements. The below code provides an example of a `for` loop that prints out each key/value pair in a dictionary:

```
menu = {'cookie': 2,'croissant': 5}
for item in menu:
  print("key:", item, " value:", menu[item])
Output[1]: key: cookie  value: 2
Output[2]: key: croissant  value: 5
```

Notebook 3: FA24_Notebook_3_DataStructures.ipynb

- **Tuples**: A tuple stores items enclosed in parentheses and separated by commas. It can store anywhere from zero to millions of items, but these items cannot be changed after the tuple is initialized. For example, `('Coffee', 'Small')` is a tuple that stores two pieces of information. Items in a tuple can be accessed using their index number.
- **Dictionaries**: A dictionary is a data structure that can hold many pairs of what are called keys and values. In a dictionary, each value is stored in relation to a descriptive key forming a key/value pair. For example, the dictionary `{'cookie': 2,'croissant': 5}` stores the prices of two items on a menu. In this case, the

key `'cookie'` maps to the value `2` and the key `'croissant'` maps to the value `5`.

- **Lists**: A list stores items enclosed in brackets and separated by commas. Like tuples, lists can store zero to many items. Unlike tuples, items in lists can be changed after the list is initialized. For example, `[85, 89, 81]` is a list that stores three pieces of information. Items in a list can be accessed using their index number.
- **Sets**: A set stores items enclosed in curly braces and separated by commas. Sets can store many items, but cannot contain duplicates. For example, `{"item1", "item2"}` is a set that stores two pieces of information. The items in a set are unordered so they cannot be accessed using index numbers.

Notebook 4: FA24_Notebook_4_Conditionals&AdvancedFunctions.ipynb

- **Conditional execution**: With conditional execution, a piece of code only executes when certain conditions are met.
- **Boolean values**: A boolean value is used to say whether something is true or false. While other data types like strings (`'1700 butterflies!'`, `'Hello World'`) and floats (`1700.5`, `1700.0`) might have lots of different values, boolean data types can only be `True` or `False`. For example, the assignment statement `python_is_fun = True` initializes a boolean variable with the value `True`.
- **Boolean expressions:** A boolean expression is an expression that is either `True` or `False`. For example, the boolean expression `'cat' == 'dog'` determines whether the string `'cat'` is the same as the string `'dog'`. Since the strings are different, the statement is `False`. There are also other comparison operators you can use to compare different values. For example, the boolean expression `15 > 10` is `True`.
- **Logical operators**: There are three logical operators in Python: `and`, `or`, and `not`. The semantics of these operators is similar to their meaning in English. For example, `x > 0` and `x < 10` is `True` only if `x` is greater than `0` and less than `10`. However `x > 0 or x < 10` is `True` if either of the conditions is `True`. So, this would be `True` as long as `x` is greater than `0` **or** less than `10`. Finally, the `not` operator negates a boolean expression, so `not (x > y)` is `True` if `x > y` is `False`; that is, if `x` is less than or equal to `y`. One more useful operator is `in`. The `in` operator is another operator that will return a boolean result of either `True` or `False`. It asks whether one piece of data is in another. For example, the `in` operator can be used to determine whether a certain key is in a dictionary.
- **Conditional Statements**: Conditional statements are used to change the behavior of a program based on whether a condition is `True` or `False`. Python has three types

of conditional statements: `if`, `else`, and `elif`. An `if` statement executes code if a condition is `True`. It can be paired with an `else` statement that executes different code if the condition is `False`. An `elif` statement, which is short for "else if," can be placed between an `if` and an `else` statement and allows us to create a list of possible conditions where one (and only one) action will be executed.

- **Combining loops and conditionals**: Combining loops and conditionals allows us to do different things to each item in a collection of items depending on its value.
- **Using functions with tuples, dictionaries, and conditional statements**: Much of what we've learned about conditionals, `for` loops, tuples, and dictionaries can help us to write complex functions and programs that can iterate and execute tasks based on different inputs. Writing a function that takes tuples or dictionaries (or both) as the input and integrates conditional statements in a loop can help us perform different tasks for each item in the data structure. For example, the below function combines two dictionaries and calculates the approximate inches of snow at each location for each month. For further explanation of this function please see Notebook 4.

```
# Initialize the input dictionaries.
avg_snow = {"Boston": 2,  "Brookline": 1,  "Cambridge": 3}
month_cold_days = {"December": 20,  "January": 31,
"February": 15}
# Define the function
def approximate_snow(snow, months):
     month_approx_inches = {}
     for location in snow:
          for month in months:
               month_approx_inches[(location, month)] =
               snow[location] * months[month]
     return month_approx_inches
# Call the function
snowfall_by_month = approximate_snow(avg_snow,
month_cold_days)
Output[1]: {('Boston', 'December'): 40, ('Boston',
'February'): 30, ('Boston', 'January'): 62, ('Brookline',
'December'): 20, ('Brookline', 'February'): 15,
('Brookline', 'January'): 31, ('Cambridge', 'December'):
60, ('Cambridge', 'February'): 45, ('Cambridge',
'January'): 93}
```