Author: Cole Cummins

An Intro to Python

Using Visual Studio Code

Visual Studio Code, or VS Code, is the development tool recommended for this course. If other tools are preferred, go ahead and use those, however it is highly recommend to use VS Code. VS Code has a built in terminal, git support, debugging, and a wide range of extensions.

Useful VS Code shortcuts:

`Ctrl + n`	Creates a new blank text file
`Ctrl + s`	Saves the current file
`Ctrl + ~`	Opens the terminal inside VS Code
`Ctrl + ,`	Opens VS Code settings

Other shortcuts can be found on the VS Code website. VS Code settings can be edited by copying and pasting the settings to you want to change into the user settings panel on the right hand side.

Hello, World!

An essential first program in every language is "Hello, World!". Copy the following code into a new blank file in VS Code. Before writing any code into the file, make sure to save the file as `helloworld.py.` I will talk about snippets of the code afterward.

```
1       def main():
2               print(“Hello, World!”)
3
4       if __name__ == “__main__”:
5               main()
```

Line `1` is a function definition of the function `main()`. Main functions will be discussed in a following section. Line `2` is indented and includes a `print()` function. Print functions allow the user to output to the terminal, with the text inside the `print()` function being outputted. Lines `4` and `5` must always be included whenever there is a main method within a file.

To execute the file, open the terminal inside VS Code and go to the same directory as `helloworld.py`. To change directories, simply use the command `cd <directory_name>.`

Using `cd ..` goes up a directory. So, if `helloworld.py` is saved on the desktop, simply type `cd desktop`. To check if a file is in a certain directory, use the command `ls` to list the contents of a directory. Execute the file by typing `python helloworld.py` into the terminal.

Variables

A variable, much like an algebraic variable, stores information. A variable could be a string, or a piece of text, an integer, a boolean, an array, etc. Type does not have to be specified for variables in python. For example:

An integer:

`count = 0` `count += 1` `count` is now equal to: `1`

A string:

`str = “”` `str += “a”` `str += “b”`

`str` is now equal to `“ab”` `str[0]` is equal to `“a”`

A list:

`list = [4, 5, 6]` `list += [7, 8]` `list[0]` is equal to `4`

`list` is equal to `[4, 5, 6, 7, 8]`

A boolean:

`bool = True`

Functions and Main methods

Functions

A function is a method with a specific purpose, where the user may pass in any variables to the function and the function returns a value. For example, in a `print()` function such as `print(“Hello, World!”)`, the string `“Hello, World!”` is being passed into the function. Another simple example of a function is `three_n_plus_one()`

```
def three_n_plus_one(n):
      result = (3 * n) + 1
      return result

print(three_n_plus_one(5))
```

Line `1` is the function definition of `three_n_plus_one()`, where `n` is a number being passed into the function. Line `3` is the function returning the result of (3 * n) + 1. Line `5` is the result being printed to the terminal.

As you may notice, a colon follows the function definition on line `1`. All function definitions must have a colon. Additionally, lines `2` and `3` are indented. Python will only recognize lines of code as being part of a function if those lines are indented.

Main Methods

A main method, as used in `helloworld.py` is prioritized above all else, and generally main methods are used to call other functions and deal with user input and output. If a main method is included in a file, the following lines must always be included in the bottom of the file

```
if __name__ == “__main__”:
      main()
```

If Statement

Logical structures are used with boolean statements. Booleans have two states, either `True` or `False`. The if statement is the basic logical structure in python and works as follows.

```
if <a boolean statement>:
      <if boolean statement is True, execute this code>
else:
      <if boolean statement is False, execute this code>
```

If statements, just like function definitions, have a colon following the boolean statement and indented lines determine what code is inside the if statement. If statements can also have an optional else statement to execute code if the boolean statement is `False`. A simple example:

```
if count == 10:
      count = count * 2
else:
      count = count/2
```

And, Or, Not, and operators

As I discussed earlier booleans are either `True` or `False`. The three main logical operators are `and, or, not`. Here are the boolean equivalents of each of these operators:

```
and
      False and False                     False
      True and False                      False
      True and True                       True

or
      False or False                      False
      True or False                       True
      True or True                        True

not
      not True                            False
      not False                           True
```

Python also has comparison operators for checking if two things are equal, as well as less than and greater than, and greater than or equal to and less than or equal to. Comparison operators:

Equal to:

```
==            ex. str == “hello!”        num == 10
```

Less than and greater than:

```
<     >     >=     <=      ex. num > 10      num <= 20
```

CodingBat

Warmup1-

```
sleep_in()                    monkey_trouble()
sum_double()                  parrot_trouble()
makes10()                     near_hundred()
```

Challenge!

Logic1-

```
cigar_party()                 sorta_sum()
date_fashion()
```

Strings

Strings are pieces of text that can be manipulated as in `“Hello, World!”` in `helloworld.py`. An empty string would be initialized as “”.

Single characters within strings can be accessed with `<string_name>[<index>]` with the index being the position of the specific character. Positive string positions are as follows:

```
str = “Hello!”
“H”   “e”   “l”   “l”   “o”   “!”
 0     1     2     3     4     5
```

For example, if I wanted to access the “o” in “Hello!” I would simply use `str[4]`. It should be noted that specific characters in a string cannot be changed once a string is created. Strings in python also support negative string positions starting from the back. For example:

```
str = “Hello!”
“H”   “e”   “l”   “l”   “o”   “!”
-6    -5    -4    -3    -2    -1
```

A useful function for strings is the function `len()`, which returns the length of the given string as an integer. For example, `len(“Hello!”)` would be equal to 6.

Substrings are another useful feature in python. A substring is a piece of a given string. Getting substrings for strings is `<string_name>[<start_index>:<end_index + 1>]`. For example:

```
str = “Hello!”
str[:2]          str[1:3]          str[3:]
“He”             “el”              “lo!”
```

If you notice in the 1st and 3rd examples, the index 0 is omitted from the first substring and the index 6 is omitted from the second. If a start or end index is not included in the substring, python assumes that the beginning or end of the string is the other index.
Python strings can also be appended and repeated.

Appending:
```
“abc” + “def”
“abcdef”
```

Repeating:
```
“abc” * 3
“abcabcabc”
```

Other useful string operations and functions can be found in the *`Python Toolkit`*

CodingBat

```
String1-
    hello_name()                    make_abba()
    first_two()                     extra_end()
    first_half()
Warmup2-
    string_times()                  front_times()
Warmup1-
    missing_char()                  front_back()
```

Lists

A list is an ordered collection of items, be it integers, strings, booleans, or even other arrays. A single list in python can hold multiple different kinds of types, ei strings and integers, however it is recommended you keep lists of the same type to avoid errors. An empty list is initialized as `list = []`.

A lot of the syntax for lists is the same as syntax for strings. For example, referencing single objects inside lists for both a positive index or a negative index is the same for lists as it is for strings. The function `len()` works for lists as it does for strings. Sublist syntax is the same as substring syntax. Lists can also be appended and repeated, like strings

```
list = [4, 10, 7, 3, 8]
list[0]          list[-1]         list[2]
4                8                7

list[1:3]        list[:3]         list[2:]
[10, 7]          [4, 10, 7]       [7, 3, 8]

[“cat”] + [“dog”]                 [“wow”] * 3
[“cat”, “dog”]                    [“wow”, “wow”, “wow”]
```

CodingBat

```
List1-
    make_pi()                       first_last6()
    same_first_last()               sum3()
    reverse3()                      sum2()
```

Other useful list operations and functions can be found in the *Python Toolkit*

While Loops

A while loop is another logical structure similar to an if statement. A while loop loops around the same section of code until its boolean statement is no longer true or the loop is forcefully exited.

```
while <a boolean statement>:
      <execute this code as long as the boolean statement is True>
```

```
count = 0
while count < 10:
      print(count)
      count = count + 1
```

In the example above, a count starts at 0, and loops and prints out each number until it reaches 10, where the loop exits. As with if statements, a colon must be placed after the boolean statement, and lines must be indented to determine what is included in the while loop.

Two ways of controlling loops are `break` and `continue.` `break` will cause the while loop to immediately exit and not continue looping. `continue` will cause the loop to skip over the current iteration and move on to the next one. For example:

```
count = 0
while True:
      if count >= 10:
            break
      if count == 6:
            continue
      print(count)
      count = count + 1
```

In the example above, a count starts at 0, and loops and prints out each number except for the number 6 (which is unlucky), and then exits once it reaches the number 10.

Assignment

Write a function called `print_odds(n)` that takes in a number `n` and prints every odd number from 1 to `n`

For Loops

A for loop is another logical structure much like a while loop, only much more useful. For loops are very good at traversing lists and strings. There are two forms of for loop, the for-each loop, and the for-range loop.

For-Each:

```
for <element> in <list or string>:
      <execute some code>
```

The for-each loop will loop through each element in either a list or string, and execute some code for each element. For example:

```
list = [4, 5, 6]
sum = 0
for num in list:
      sum = sum + num
```

The above example loops through each number in the list and adds it to the sum. Much like a while loop and if statement, a colon and indents are used to determine what code is inside the for loop.

For-range

```
for i in range(<ending_index + 1>):
      <execute some code>
```

The for-range loop is by far the most useful logical structure, useful for looping through strings and lists. The `i` in the for range loop represents a sequence of numbers starting at 0 and ending at the ending index.

CodingBat

```
      String2-
            double_char()                       cat_dog()
            count_code()                        xyz_there()
      List2-
            count_evens()                       sum13()
            sum67()
```

Challenge!

```
      Logic2-
            make_bricks()                       make_chocolate()
```