My Answers to Automate the Boring Stuff with Python

Chapter 1

1. Which of the following are operators, and which are values?

```
* <- operator
'hello' <- not operator
-88.8 <- not operator
-  <- operator
/ <- operator
+ <- operator
5 <- not operator
```

2. Which of the following is a variable, and which is a string?

```
spam <- variable
'spam' <- String
```

3. Name three data types.

A: *integer, float, string*

4. What is an expression made up of? What do all expressions do?

*An expression is made up of **values** and **operators**. Expresssions **evaluate** the values and operators down to a single value.*

5. This chapter introduced assignment statements, like spam = 10. What is the difference between an expression and a statement?

Assignment statements link a variable a value. An expression requires the use of operators, can use multiple vairables in the statement, and does not need to result in a variable being assigned.

6. What does the variable bacon contain after the following code runs?

```
bacon = 20
bacon + 1
```

A: *20, because* `'bacon'` *was not assigned the result of* `'bacon + 1'`

7. What should the following two expressions evaluate to?

```
'spam' + 'spamspam'
'spam' * 3
```

A: `'spamspamspam'` and `'spamspamspam'`

8. Why is eggs a valid variable name while 100 is invalid?

You cannot begin a variable with a number. 'eggs' contains no illegal characters nor does it start with a number

9. What three functions can be used to get the integer, floating-point number, or string version of a value?

`str(), int(), and float() functions`

10. Why does this expression cause an error? How can you fix it?

```
'I have eaten ' + 99 + ' burritos.'
```

A: *Cannot concatenate strings and integers either* `'I have eaten ' + '99' + ' burritos.'` *or* `'I have eaten ' + str(99) + ' burritos.'`

Chapter 2

1. What are the two values of the Boolean data type? How do you write them?

'*True*' *and* '*False*'

2. What are the three Boolean operators?

`and`, `or`, *and* `not`

3. Write out the truth tables of each Boolean operator (that is, every possible combination of Boolean values for the operator and what they evaluate to).

Easy to do but too much work.

4. What do the following expressions evaluate to?

```
(5 > 4) and (3 == 5) <- False
not (5 > 4) <- False
(5 > 4) or (3 == 5) <- True
not ((5 > 4) or (3 == 5)) <- False
(True and True) and (True == False) <- False
(not False) or (not True) <- True
```

5. What are the six comparison operators?

`<, >, <=, >=, ==, !=`

6. What is the difference between the equal to operator and the assignment operator?

Equal operator is two equal signs, '=', *while the assignment operator is one equal sign,* '='.

7. Explain what a condition is and where you would use one.

A logical test that determines if a control flow statement is executed. You would use one in `if` , `elif` `while` , *and* `do{} while` *statments as well as functions that use a boolean condition as input*

8. Identify the three blocks in this code:

```
spam = 0 <- Start of first block
if spam == 10:
    print('eggs')
    if spam > 5:
        print('bacon') <- second block
    else:
        print('ham') <- third block
    print('spam')
print('spam') <- end of first block
```

Jay's notes: *the way the author identifies blocks is not the same way most other people would look at blocks. Blocks would normally be every place where the lines are indented in Python. I wouldn't worry too much about understanding this section anyways. It'll just come naturally to you over time.*

9. Write code that prints Hello if 1 is stored in spam, prints Howdy if 2 is stored in spam, and prints Greetings! if anything else is stored in spam.

A:

```
if(spam == 1):
  print("Hello")
elif(spam == 2):
  print("Howdy")
else:
  print("Greetings!")
```

10. What can you press if your program is stuck in an infinite loop?

```
ctrl-C
```

11. What is the difference between break and continue?

'break' stops the local loop's execution. 'continue' stops the current iteration and continues on to the next one

12. What is the difference between range(10), range(0, 10), and range(0, 10, 1) in a for loop?

Nothing

13. Write a short program that prints the numbers 1 to 10 using a for loop. Then write an equivalent program that prints the numbers 1 to 10 using a while loop.

```
for i in range(1, 11):
    print(i)

j = 1
while(j <= 10):
```

```
print(j)
j = j + 1
```

14. If you had a function named bacon() inside a module named spam, how would you call it after importing spam?

`spam.bacon()` *or* `bacon()` *if import was done as `import * from spam`

Chapter 3

Q:

1. Why are functions advantageous to have in your programs?

A:

Because you can group code to be executed multiple times so you don't have to duplicate it throughout your program.

Q:

2. When does the code in a function execute: when the function is defined or when the function is called?

A:

When the function is called.

Q:

3. What statement creates a function?

A:

def

Q:

4. What is the difference between a function and a function call?

A:

The function call is a reference to the previously defined function. The function call contains values for the function parameters and uses the result of the function being processed useing those parameters.

Q:

5. How many global scopes are there in a Python program? How many local scopes?

A:

There is only one global scope. There are as many local scopes as there are code blocks.

Q:

6. What happens to variables in a local scope when the function call returns?

A:

They are destroyed

Q:

7. What is a return value? Can a return value be part of an expression?

A:

A return value is what the function call becomes when it completes, assuming there is a return statement in the definition of the function. A return value can be part of a statement. EX: """ def functionThatIncrementNumber (number): return number + 1

someVariable = 5 * functionThatIncrementNumber(1) print(someVariable) """ Console Output: 10

Q:

8. If a function does not have a return statement, what is the return value of a call to that function?

A:

None

Q:

9. How can you force a variable in a function to refer to the global variable?

A:

Using the keyword 'global'.

Q:

10. What is the data type of None?

A:

NoneType

Q:

11. What does the import areallyourpetsnamederic statement do?

A:

Links your current program with a module named "areallyourpetsnamederic". If there are functions or classes in areallyourpetsnamederic, you can now use them.

Q:

12. If you had a function named bacon() in a module named spam, how would you call it after importing spam?

A:

spam.bacon()

Q:

13. How can you prevent a program from crashing when it gets an error?

A:

Catch the exception that was generated when that error was made.

Q:

14. What goes in the try clause? What goes in the except clause?

The code that may generate an error to be caught is in the try clause. The code to handle when a specific error occurs goes in the except clause. There should be a specific except clause for each error that could be generated.

Chapter 4

Q:

1. What is []?

A:

An empty list.

Q:

2. How would you assign the value 'hello' as the third value in a list stored in a variable named spam? (Assume spam contains [2, 4, 6, 8, 10].)

A:

spam[2] = "hello"

For the following three questions, let's say spam contains the list ['a', 'b', 'c', 'd'].

3. What does spam[int(int('3' * 2) // 11)] evaluate to

The fourth element of `spam`, `spam[3]`, *which would evaluate to 'd' using the previous assignment of spam.*

4. What does `spam[-1]` evaluate to?

Negitive indices start from -1 and the last index in the array. Using the previously defined spam vairable, `spam[-1] == 'd'`

5. What does spam[:2] evaluate to?

A: `spam[:2] == ['a', 'b']`

For the following three questions, let's say bacon contains the list `[3.14, 'cat', 11, 'cat', True]`.

6. What does `bacon.index('cat')` evaluate to?*

Since index returns the first instance of a element occurance, `bacon.index('cat')` *would return 1*

7. What does `bacon.append(99)` make the list value in bacon look like?

Using `bacon == [3.14, 'cat', 11, 'cat', True]`

A: `[3.14, 'cat', 11, 'cat', True, 99]`

8. What does `bacon.remove('cat')` make the list value in bacon look like?

Using `bacon == [3.14, 'cat', 11, 'cat', True]`

A: `bacon == [3.14, 11, 'cat', True]`

9. What are the operators for list concatenation and list replication?

`The plus sign, +, and the asterisk sign, *.`

10. What is the difference between the `append()` and `insert()` list methods?

`append()` *inserts an element at the end of the list.* `insert()` *inserts an element at the specified index*

11. What are two ways to remove values from a list?

`list``.remove(`*`value`*`)` *and* `del` *`list`*`[`*`index_of_value`*`]`

12. Name a few ways that list values are similar to string values.

- *You can retrieve a character of a string or an element of a list through similar methods:* *`string`*`[`*`index_of_character`*`]` *and* *`list`*`[`*`index_of_element`*`]`
- *You can retrieve a subsection of a string or a list through similar methods:* *`string`*`[`*`begining_index`*`:`*`end_index_+_1`*`]` *and* *`list`*`[`*`begining_index`*`:`*`end_index_+_1`*`]`

13. What is the difference between lists and tuples?

Lists are mutable, as in the elements can be changed after creation. Tuples are not mutable

Jay's notes: see this StackOverflow answer and this answer for more insight on when to use lists or tuples. Don't worry if it seems to be esoteric right, the utility of each will become more obvious as you go on.

14. How do you type the tuple value that has just the integer value 42 in it?

```
(42,)
```

15. How can you get the tuple form of a list value? How can you get the list form of a tuple value?

You can convert a tuple to a list or a list to a tuple using the following built-in functions:

`tuple(`*`list_to_convert`*`)`

`list(`*`tuple_to_convert`*`)`

16. Variables that "contain" list values don't actually contain lists directly. What do they contain instead?

Memory references to those lists

17. What is the difference between `copy.copy()` and `copy.deepcopy()`?

`copy.copy()` *only copies the top level of a list to another list. If that list contains other lists,* `copy.copy()` *will only copy the references to those other lists.* `copy.deepcopy()` *will copy all values of any embedded lists.*