Python Warmup Exercises

*Python is completely object-oriented, and not "statically typed". You do not need to declare variables before using them or declare their type.

*Every variable in Python is an object.

Make sure to save all your work for each example.

Exercise 1

Python supports two types of numbers - integers and floating point numbers. (It also supports complex numbers, which will not be explained in this tutorial).

To define an integer, use the following syntax:

```
myint = 7
print(myint)
```

Exercise 2

To define a floating point number, you may use one of the following notations:

```
myfloat = 7.0
print(myfloat)
myfloat = float(7)
print(myfloat)
```

Exercise 3

Strings are defined either with a single quote or a double quotes.

```
mystring = 'hello'

print(mystring)
mystring = "hello"
print(mystring)
```

Exercise 4

The difference between the two is that using double quotes makes it easy to include apostrophes (whereas these would terminate the string if using single quotes)

```
mystring = "Don't worry about apostrophes"
print(mystring)
```

Exercise 5

There are additional variations on defining strings that make it easier to include things such as carriage returns, backslashes and Unicode characters. These are beyond the scope of this tutorial but are covered in the Python documentation.

Simple operators can be executed on numbers and strings:

```
one = 1
two = 2
three = one + two
print(three)

hello = "hello"
world = "world"
helloworld = hello + " " + world
print(helloworld)
```

Exercise 6

Assignments can be done on more than one variable "simultaneously" on the same line like this

```
a, b = 3, 4
print(a,b)
```

Exercise 7

Assignments can be done on more than one variable "simultaneously" on the same line like this

```
# This will not work!
one = 1
two = 2
hello = "hello"

print(one + two + hello)
```

Exercise 8

A list is a linear data type. It can contain any type of variable, and they can contain as many variables as you wish. Lists can also be iterated over in a very simple manner. Here is an example of how to build a list.

```
mylist = []
mylist.append(1)
mylist.append(2)
mylist.append(3)
print(mylist[0]) # prints 1
print(mylist[1]) # prints 2
print(mylist[2]) # prints 3
```

Exercise 9

Accessing an index which does will generate an exception (an error).

```
mylist = [1,2,3]
print(mylist[10])
```

Exercise 11

In this exercise, you will need to add numbers and strings to the correct lists using the "append" list method. You must add the numbers 1,2, and 3 to the "numbers" list, and the words 'hello' and 'world' to the strings variable.

You will also have to fill in the variable second_name with the second name in the names list, using the brackets operator []. Note that the index is zero-based, so if you want to access the second item in the list, its index will be 1.

```
numbers = []
strings = []
names = ["John", "Eric", "Jessica"]
```

Exercise 12

Python includes syntax for slicing sequences into pieces. Slicing lets you access a subset of a sequence's items with minimal effort. The simplest uses for slicing are the built-in types list, str, and bytes. The basic form of the slicing syntax is somelist[start: end], where the start is inclusive and the end is exclusive. When slicing from the start of a list, you should leave out the zero index to reduce visual noise. When slicing to the end of a list, you should leave out the final index because it's redundant. Using negative numbers for slicing is helpful for doing offsets relative to the end of a list. All of these forms of slicing would be clear to a new reader of your code. There are no surprises, and I encourage you to use these variations.

```
a = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
print('First four:', a[:4])
print('Last four: ', a[-4:])
print('Middle two:', a[3:-3])
```

Exercise 13

Python includes syntax for slicing sequences into pieces. Slicing lets you access a subset of a sequence's items with minimal effort. The simplest uses for slicing are the built-in types list, str, and bytes. The basic form of the slicing syntax is somelist[start: end], where the start is inclusive and the end is exclusive. When slicing from the start of a list, you should leave out the zero index to reduce visual noise. When slicing to the end of a list, you should leave out the final index because it's redundant. Using negative numbers for slicing is helpful for doing offsets relative to the end of a list. All of these forms of slicing would be clear to a new reader of your code. There are no surprises, and I encourage you to use these variations.

```
a = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']

print(a[:] )
print(a[:5]
print(a[:-1]
print(a[4:]
print(a[-3:]
print(a[2:5]
print(a[-3:-1]
print(a[-3:-1]
```

Exercise 14

Tuples are identical to lists in all respects, except for the following properties. Tuples are defined by enclosing the elements in parentheses (()) instead of square brackets ([]). Tuples are immutable.

```
t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')
print(t)

print(t[0])
print(t[1])
print(t[1::2])
```

Exercise 15

Everything you’ve learned about lists—they are ordered, they can contain arbitrary objects, they can be indexed and sliced, they can be nested—is true of tuples as well. But they can’t be modified.

```
t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')
t[2] = 'Bark!'
```

Exercise 16

Dictionaries are mutable objects in Python, they can grow and shrink as needed. It can also contain another dictionary or any other objects.

```
MLB_team = {
      'Colorado' : 'Rockies',
      'Boston'   : 'Red Sox',
      'Minnesota': 'Twins',
      'Milwaukee': 'Brewers',
      'Seattle'  : 'Mariners'
}


print(MLB_team)
print(MLB_team['Boston'])
```

Exercise 16

Exercise 16 Dictionaries are mutable objects in Python, they can grow and shrink as needed. It can also contain another dictionary or any other objects.

```
MLB_team = {
      'Colorado' : 'Rockies',
      'Boston'   : 'Red Sox',
      'Minnesota': 'Twins',
      'Milwaukee': 'Brewers',
      'Seattle'  : 'Mariners'
}
```

```
MLB_team['Kansas City'] = 'Royals'
print(MLB_team)
```

Exercise 17

You have already become familiar with many of the operators and built-in functions that can be used with strings, lists, and tuples. Some of these work with dictionaries as well. For example, the in and not in operators return True or False according to whether the specified operand occurs as a key in the dictionary.

```
MLB_team = {
      'Colorado' : 'Rockies',
      'Boston'   : 'Red Sox',
      'Minnesota': 'Twins',
      'Milwaukee': 'Brewers',
      'Seattle'  : 'Mariners'
}

print('Milwaukee' in MLB_team)
print('Toronto' in MLB_team)
print('Toronto' not in MLB_team)
print(MLB_team.items())
print(MLB_team.keys())
```