Python Cheat Sheet

Whitespace

If you're familiar with Javascript, one of the things that you'll notice is that Python doesn't use a mix of braces and semicolons to section off statements. Instead, Python uses whitespace (either a tab or four spaces) to separate out statements. The general rule is that if a statement ends in a colon, you need to indent.

Variables

A variable in Python is created by assigning it. Assignments look like:

```
a = 2
b = 'some text'
```

In the above examples, the variable names are a and b, though they can be anything you want them to be, as long as they don't contain a space or start with a number. So, yes, you can have:

```
my_totally_awesome_variable_name_12 = 2
```

The = sign is important here, it's what tells Python, 'Hey, this name equals this thing'. Once you assign something a variable, you can refer to it by name and change the value to your heart's content. For example, if you want to add 100 to a, you just write:

```
a = a + 100
```

And now a will equal 102.

Text, i.e. Strings

Text in Python is called a string. You specify that something is a bit of text by simply putting quotation marks, either single or double, around the text you want to use. So, to work with the glyph name aacute, you write:

```
'aacute'
```

Numbers

A number in Python is simply declared by writing out the number, no quotation marks at all.

True/False

Sometimes you need to be able to find out if something is True or False. Is the width of your glyph greater that 500? Is the name 'myGlyph'? For this, Python has booleans, which is a fancy name for **True** or **False**. To find out if something is True, you use the boolean operators which are:

operator	*example*
== equals	2 == 2
!= not equals	1 != 2
< less than	1 < 2
> greater than	2 > 1
<= less than or equal	2 <= 2
>= greater than or equal	3 >= 2

One can use these operators to find out if anything is equal or not equal to another thing. Each example will either return **True** or **False**.

Tupple

Besides being a funny word, a tupple is a way in Python to represent a an ordering of things that won't change. The best example of this is point coordinates. If you want to represent the x and y values for a point on your glyph, you would use a tupple and it would look like:

```
my_point = (10, 40)
```

Tupples are written in between parenthesis, and each item of the tupple is separated by a comma.

The special thing about tupples is that once you create one, the values of the tupple can't change. To change the values of a tupple, you have to make a new one. To get and change a value of a tupple you write:

```
my_new_tupple = (my_point[0] + 10,
my_point[1] + 10)

my_new_tupple will now be (20, 50)
```

If/Else

Likewise, one can then use if something is true in a if/elif/else statement. An example is:

```
if first == True:
      print 'first!'
elif second == True:
      print 'second!'
else:
      print 'none :('
```

Python Cheat Sheet

You might be wondering what elif stands for, it's shorthand for 'else if'. If the if statement is not True, you can have any number of elif statements to test things until your final else statement which will only run if nothing has matched to true.

List

A list is a way of storing a sequence of items that you can look at one by one. Lists are constructed by enclosing a comma separated list by brackets, i.e.:

```
a_list = [1, 2, 3, 4]
```

One can add items to a list that has been named by calling the append method on a list, i.e.:

```
a_list.append(5)
```

Counting

Counting in Python starts with 0, so if you want the first thing, you ask for the 0 thing. This is easily the most confusing bit of computer programming. An example:

```
my_list = [1, 2, 3, 5]
print my_list[1]
  2
print my_list[0]
  1
```

If you want the first item of your list, you have to ask for 0, not 1.

Dictionary

Like a list, a dictionary is a way of storing information, but instead of a straight list of things, a dictionary is a mapping of a 'key' to a 'value'. Many things in typeface design can be thought of this way. Say, the 'key' of (A, V) has the value of -100 in a kerning dictionary. Dictionaries are created like so:

```
a_dictionary = {'first key': 'first
value', 'second key': 'second value',
'third key': 'third value'}
```

The important thing to note is that there is a colon between a key and it's value, and that each key/value pair is separated by a comma. The above example is all strings, but a dictionary can be a collection of anything.

To add an item to a exiting dictionary, you write this:

```
a_dictionary['fourth key'] = 'fourth
value'
```

This form is simply the name of the dictionary followed by the name of the new key in between brackets with what the value is after an equal sign.

If you want to get a specific value from a dictionary, you write:

```
a_dictionary['first key']
```

Python will return the value for that key.

For Loops

Lists and dictionaries are meant to be 'looped' over. The way to do this in Python is a for loop. For example, using the list from above:

```
for item in list:
      print item
```

The structure that is important here is the for and the in. Item and list are variable names, and can be anything you want them to be. What happens is that Python will look at each item in your list one by one, setting the item variable name to the item in the list. You can then do things with the item, above just printing it, and then do the same thing to the next item in the list, and so on.

To get a list of items from a dictionary, you write:

```
for item in my_dictionary.keys():
      print item
```

Dictionaries have a couple of special methods to get a list of their keys or values, that is what we used above to get a list of all the keys in a dictionary. These methods are:

```
my_dict.keys()    List of keys
my_dict.values()  List of values
my_dict.items()   List of (key, value)
```

These methods come in handy when working with dictionaries in a for loop.

Python Cheat Sheet

Print

The print statement is the most valuable tool you have for getting feedback on what is going on in your script. Anytime you need to know what a variable is set to, or want to give some feedback you can write:

```
print ‘Test!’
print my_variable
```

Python will then return the string or value for your variable.