Intro to Python

Quick-Reference

Type@Cooper

Concepts and puzzle-pieces

White Space

Designers love Python because white space is meaningful in the world of Python! Python doesn't use a mix of braces and semicolons to separate statements. Instead, Python uses whitespace (either a tab or four spaces). The general rule is that if a statement ends in a colon, you need to indent.

Think of white space as a means of creating hierarchy, similar to how the title of this page is at the top and separated by a bunch of white space.

```
for glyph in font:
    do_something(glyph)

    for contour in glyph.contours:
        do_something_else(contour)

        for pt in contour.points:
            change_point(pt)

    print('Did stuff to a glyph!')
```

Variables

A variable in Python is a way of storing or assigning information for later.

Variable names can be anything you want them to be, as long as they don't contain a space or start with a number.

Variable *values* can be anything, a number, string, list, function etc.

The = sign is important, it's what tells Python, 'this name equals this thing'. Once create a variable and assign it a value, you can refer to it by name and change the value over and over again.

```
a = 2
a = a + 100  # Change it all you want!

b = 'Some text'

my_name = 'Sue'

super_cool_variable_name = 3
```

Print

```
print('Test 1!')

my_variable = 'Test 2!'
print(my_variable)
```

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, use print(), and Python will then print out the string or value for your variable.

Strings

```
print('Hello, world.')

glyph_name = 'Agrave'

weird_string = '29509undsinht920nk'
```

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.

Numbers

```
print(3)

your_favorite_integer = 64

cool_negative_float = -0.45
```

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

Two basic forms a number can take are:

- an integer (e.g. -2, 0, 1, 10)
  or
- a float (e.g. -2.1, 0.62, 1.2, 10.0)

Counting

```
my_list = [1, 2, 3, 5]
print(my_list[1])  # Prints 2

# If you want the first item of your
# list, you have to ask for 0, not 1.
print(my_list[0])  # Prints 1
```

Counting in Python starts with 0, so if you want the first thing, you have to ask for the 0(th?) thing.

This is easily the most confusing bit of computer programming. You can think of things like a progress bar (nothing done yet is 0, not 1) or a distance measurement (beginning point is 0, not 1).

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 its name ‘Agrave’? 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 operations, which are:

- == equals
- != not equals
- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to

Interestingly enough, True is essentially equal to 1, and False is essentially equal to 0.

```
print(2 == 2)      # Will print True

print(2 != 2)      # Will print False

print(1 < 2)       # Will print True

print(1 > 2)       # Will print False

print(1 <= 2)      # Will print True

print(2 >= 2)      # Will print True

# Bonus
print(1 == True)   # Will print True
```

If Statements

To check if something is True or False, you can use an **if/elif/else** statement.

'elif' is a shorthand for 'else if'. After an 'if' statement is written, you can follow it up with any number of 'elif' statements to test things until your final 'else' statement which will only run if nothing previously caught on.

You need an initial 'if' statement for any 'elif' or 'else' statements to be valid.

```
course_name = 'Intro to Python'

if course_name == 'Pottery 101':
    print('We're kiln it!')
elif course_name == 2:
    print('Weird number title...')
elif course_name == 'Lettering':
    print('Coding's involved?')
else:
    print('Nope, it's', course_name)

# Should print:
# Nope, it's Intro to Python
```

```
# Ways of storing info

## Lists
['a', 'b', 'c']

## Tuples
('a', 'b', 'c')

## Dictionaries
{0: 'a', 1: 'b', 2: 'c'}
```

Lists

```
students = ['Tim', 'Sally', 'Jia']
print(students[0])  # Prints 'Tim'

cool_numbers = [0, 28, 2048]
cool_numbers.append(12)
```

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 []

You can pick out an individual item in a list using its 'index' with a [].

You can change stuff in a list! For example, you can add items to a list that has been named by calling the **append** method on a list.

Tuples

A tuple is a way in Python to represent a an ordering of things. They're like lists, but they'll never change! If you want to change the values of a tuple, you have to make a new one.

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 tuple.

Tuples are written in between parentheses (), and each item of the tuple is separated by a comma.

```
pt = (10, 40)

new_tuple = (pt[0] + 10, pt[1] + 10)
print(new_tuple)  # Is now (20, 50)


# ————————
# Tuples can have any number of items
useless_tuple = (2, 3, 1, 'hello')
```

Dictionaries

Like lists and tuples, 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. for example, the 'key' of (A, V) might have the value -100 in a kerning dictionary.

There is always a colon between a key and it's value, and each key/value pair is separated by a comma.

You can check or change values of keys, and you can add keys.

```
my_dictionary = {
        'first key': 'first value',
        'second key': 'second value',
        'third key': 'third value'
                }
# ———

apple_ownership = {
        'Jaclyn': 24,
        'Tim': 2,
        'Steve': 1
                }

print(apple_ownership['Tim'])  # Prints 2

# Give Tim another apple!
apple_ownership['Tim'] = 3
```

Putting it all together and doing things

For Loops

For loops are powerful and useful!

Databases like Lists and dictionaries are meant to be ‘looped’ over. The way to do this in Python is a for loop. The **for** and the **in** are crucial.

When looping over a list, Python will look at each item in your list one by one, setting the chosen variable name to the item in the list. You can then do things with the item, or just print it, and it repeats to the next item in the list, and so on, until the list is all over.

Dictionaries have a couple of special methods to get a list of their keys or values, which is what we use to get a list of all the info in a dictionary.

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

```
# List looping
# item and my_list are variable names,
# and can be anything you want them to be
my_list = ['a', 'b']
for item in my_list:
    print(item)

# Dictionary looping
my_dict = {'Jaclyn': 24, 'Tim': 2}
for key, value in my_dict.items():

    print(key, 'has', value)
```

Functions (aka Methods)

Functions are ways of grouping up and organizing sets of tasks. Pretty much anything you script can be packaged in a function. Doing this allows you to reuse those set of instructions, apply them to multiple other things, or just keep you sane and organized.

A function can be simple or complex. It can just do one simple thing, with no input at all, or it can be fed input called 'arguments' and perform actions to those arguments.

```
def say_hi():
    print('Hi.')

say_hi()

# ———

def print_list_items(a_list):
    for item in a_list:
        print(item)

list_one = ['a', 'b']
list_two = ['c', 'd']
print_list_items(my_list)
```

DrawBot

Analogues with InDesign

translate()

newPage()

text(), textBox(), formattedString()
line()
BezierPath()

rect(), oval(), polygon()

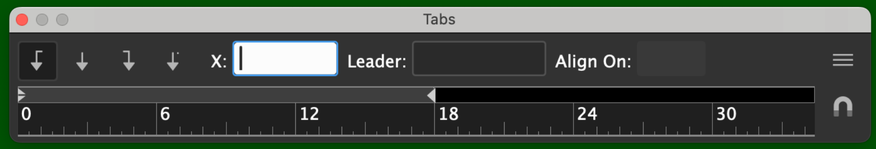
scale(), rotate(), skew()
linearGradient(), radialGradient()

imagePixelColor()

fill()

stroke()

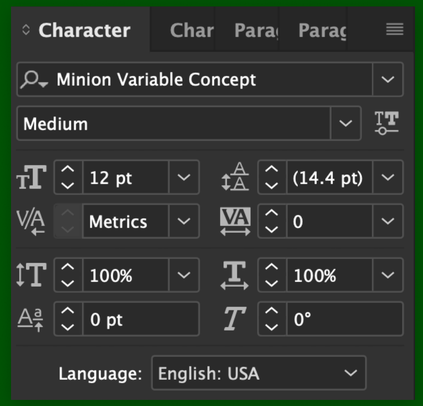
tabs()



font()

fontSize()

baselineShift()



fontVariations()

lineHeight()

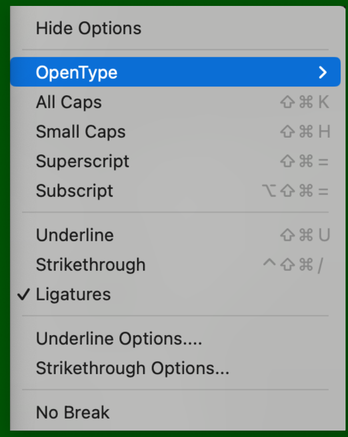
tracking()

scale()

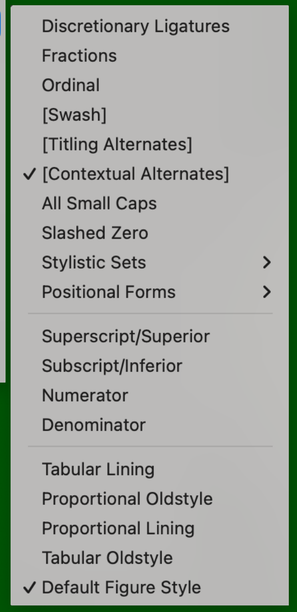
skew()

language()

openTypeFeatures()



listOpenTypeFeatures()



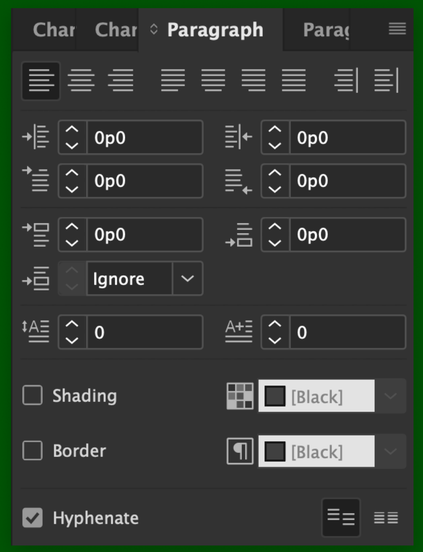
FormattedString()

.align

.indent

.firstLineIndent

.paragraphTopSpacing



.tailIndent

.paragraphBottomSpacing

hyphenation()

Resources

Scripting in DrawBot

drawbot.com

Scripting Fonts

fontparts.robotools.dev
robofont.com/documentation/tutorials/python/
glyphsapp.com/learn/

Miscellaneous

pythonfordesigners.com

```
print('Enjoy!')
```