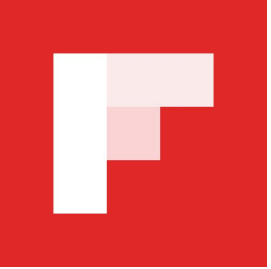

+

python™

Basics, Control Flow, & Loops

Goals for the Course

- Be able to read & write simple, elegant, and useful Python scripts
- Understand programming concepts
- Work through practice problems & build problem-solving skills
- Complete a series of weekly projects
- Have fun!!!!

Basics of Python I: Types

- **Integers**
  - Whole numbers, positive or negative
  - 1, 42, -1001, 209738964827
- **Floating point numbers (floats)**
  - A decimal number
  - 3.14
  - 26.5
  - 99.99999

- **Strings**
  - Any string of characters enclosed in single or double quotes
  - ‘x’
  - ‘Hello’
  - “my name is anna”
  - “123”

You can print any of these values to the console using `print(<value>)` e.g. `print(1), print(3.14), print('hello')`

Expressions = Value(s) + Operator(s)

Integers and Floats:

- Addition:
  - `2 + 3`
- Subtraction:
  - `1.2 - .3`
- Multiplication:
  - `2 * -3`
- Division:
  - `3//2 # == 1.5`
  - `3/2 # == 1`
- Modulus
  - `3%2 (remainder after integer division, so this returns 1)`
- Exponentiation
  - `3**2`

Strings:

- Addition (only works with two or more str):
  - `“A” + “b” # == “Ab”`
  - `“Py” + “thon # == “Python”`
  - `“Hello” + “ “ + “world”`
- Multiplication (str * int):
  - `“a”*5 # == “aaaaa”`
- Subtraction and division don’t work!

All:

- Get a value’s type:
  - `type(“apple”)`
  - `type(42)`
- Print
  - `print(“apple”)`

Variables

Think of variables as a little place in memory that you can create to store a value so that it's easily accessible later. We create variable names and assign values to them with a single equals sign, like so:

```
a = 5

my_variable = "56 cats"

fl_user_count = 2897386487248274098298
```

Variable names are case-sensitive, so `Var` is different from `var`

Conventionally, variable names begin with a lowercase letter and can contain alphanumeric characters and underscores. They cannot match Python's reserved keywords and should not match builtin names (like `int`, `float`, `str` etc.)

So these are all valid:

```
a, aVariable, a_variable, variable42
```

but these are not:

```
2var, ***var, ke$ha, print
```

Variable Operations & Practice

We can assign results of operations between values or variables to variable names.

```
numCats = 100 + 200

purple = “pur” + “ple”

purple_cats = str(numCats) + purple
```

We can increment a variable by referencing the variable itself:

```
i = i + 3

i += 3
```

It’s good practice to name your variables descriptively, so a reader (this includes you!) can more easily determine from a glance what your code does.

For example, if we’re trying to keep track of farm animals, a variable named `numberOfCows` is much more informative than a variable named `c`

First project: Write a program that prints the string “Hello World!” to the console.

Start by creating a file on the command line and opening it in your text file.

Open a new python file, by going to file -> new file or hitting cmd + n

Test your code when you’re done writing by going to run -> run module or with fn + F5

Save your file as hello_world.py in a folder of your choice file -> save as… or shift + cmd + s

Feel free to work with each other and ask questions!

Now, let’s make that program better with I/O!

We should greet people by name! We can do this by collecting user input on the command line:

```
name = input(“What is your name?: ”)

print(“Hello ” + name)
```

This offers the user to type in a string, which will be stored in the variable named `name`

So if I type in “Anna”, the string “Hello Anna” will be printed to the console.

Add this to your hello_world.py program

Now, let's make that program better with I/O!

Tip: Python has a built-in function that tells you how many characters are in your string. You can get that information by calling `len(<string>)`

```
len("anna") # = 4

name = "robert"

print(len(name)) # = 6
```

Let's add this to our `hello_world.py` program! After greeting the user, figure out how long their name is, and print out that information on the console. Your output should be something like:

```
What is your name?: Anna

Hello Anna. Your name is 4 characters long.
```

One more improvement!

Let's ask the user how old s/he is! We already know how to collect input, so this should be straightforward. Collect their age in a variable named `age`. Print to the console, telling the user how old they will be in a year.

```
How old are you?: 21

You will be 22 in one year!
```

Did you get a `TypeError` when you ran that code?

That's because `input` reads strings, not integers. `21` is not the same as `"21"`, but converting is pretty easy with `int(age)`. Once `age` is an integer, you can increment it.

In order to print this incremented number as part of the result string, you must convert it back to a string, so your program should look something like this:

```
age = input("How old are you?: ")

print("You will be " + str(int(age) + 1) + " in a year!")
```

Let's break into partners & work on "Practice Questions" (pg 28-29 or bottom of web page)

1. Operators: `*, -, /, +;` Values: `'hello', -88.8, 5`
2. `spam` is a variable name, `'spam'` is a string
3. string, integer, float
4. An expression is a programming instruction that consists of values and operators and evaluates into a single value.
5. An expression evaluates to a single value. A statement does not.
6. `bacon` still contains `20`, because the incremented value `20` was not reassigned to the variable named `bacon`
7. Both expressions evaluate to `'spamspamspam'`
8. Variable names cannot start with numbers, so `100` is invalid
9. `int(), float(), str()`
10. You cannot concatenate an int to a string without first casting the int to a string. This is a `ValueError`. So `'I have eaten ' + str(99) + ' burritos.'` works. Enclosing 99 in quotes would also work.

Introduction to Control Flow: Boolean Logic

A big part of programming is deciding whether or not to execute code based on a condition. For example, some logic we may have here at Flipboard:

If the user lives in Texas, show them a Whataburger ad. Otherwise if the user lives in California, show them an In-N-Out ad. If none of the previous conditions apply, show them a McDonald's ad.

Built into Python is another type called `bool`. A `bool` has two possible values: it can be `True` or it can be `False`.

Like the other types we've covered, `bool` can be used in expressions and stored in variables.

Acceptable uses:

```
True; False; rain = True; true = True;
```

Won't work:

```
True = True; false; true;
```

Comparison Operators

operator	meaning
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

As you can imagine, these operators resolve to boolean values depending on the two values being compared. So for an exercise, what would these values resolve to?

```
'Hello' == 'Hello'

False != False

27 < 21

23 >= 23

"Flipboard" == "flipboard"
```

Try this with variables too.

Boolean Operators

Boolean operators `and` & `or` take two boolean values and evaluate them together, providing an outcome based on the set of logical rules you see in these truth tables

a	b	a and b	a or b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Another boolean operator is called `not`. This modifies a boolean value to be the opposite. So for example if we have some variable:

```
enabled = True
```

```
not enabled # results in False
```

We can toggle between states with reassignment:

```
enabled = not enabled
```

Or save the result in another variable:

```
disabled = not enabled
```

Short exercise: Evaluate these expressions

```
(4 < 5) and (5 < 6)

(4 < 5) and (9 < 6)

(4 < 5) or (9 < 6)

(1 == 2) or (2 == 2)

2 + 2 == 4 and not 2 + 2 == 5

3 - 2 + 1 or 2 * 1 and 2
```

**Order of operations: comparison operators, then not, then and, then or

Controlling topics

A short (but relevant) program:

```
if topic == “news”:
    print(“This article is about news.”)
elif topic == “sports”:
    print(“This article is about sports.”)
elif topic == “tech”:
    print(“This article is about tech.”)
else:
    print(“I don’t care.”)
```

What does this do when the topic is “food”? Or “Sports”?

Control Flow

Terminology:

Condition: a boolean value (or an expression that evaluates to a boolean value) that determines what flow a program should take. If a computer program is a train, conditions are the railway switch.

Blocks: A group of python statements. Blocks begin at an indentation increase and end when that indentation decreases.

In the program to the right, the condition is checking whether the sky is blue. The indented print statements form two blocks.

```
if sky == blue:
    print("It must be sunny.")
    print("What a beautiful day!")
else:
    print("Crap, it's raining.")
    print("Guess I'm wfh today.")
```

if/elif/else

`if` is a keyword followed by a condition. When the program executes, the block beneath an `if` statement will only execute if the condition is `True`.

`elif` is another keyword, short for “else if”, which executes a block of code when both its condition is `True` and the previous `if` or `elif` statement is `False`

`else` indicates that the program should execute its associated block when the all of the previous `if/elif` statements are `False`. It’s the default.

```
if user == “spammer”:
    print(“disabled”)
elif user == “troll”:
    print(“trollverse”)
elif user == “vip”:
    print(“access granted”)
else:
    print(“no complaints”)
```

Program: Control the Flow of Vampires

We need to determine whether a Flipboard user is a vampire. Given a user, we know for certain that they are a vampire if at least one of the following is true: they are immortal, they are passionate about the “Nocturnal” topic, or they have muted the source “Garlic Monthly.” Write a program called vampires.py that takes a user’s name, age, favorite topic, and muted source to determine if the user is a vampire. Example outputs:

```
What is your name?: Vlad Dracula
What is your age?: 2100
What is your favorite topic?: World
Domination
What source have you muted? Garlic
Monthly

Vlad Dracula is a vampire!
```

```
What is your name?: Jonathan Harker
What is your age?: 31
What is your favorite topic?: Real Estate
What source have you muted?: The Transylvanian
Bugle

Jonathan Harker is not a vampire!
```

`while` Loops

We need a user to log in with the correct password, but we also don't want to penalize people for making typos.

While loops allow us to execute a block of code repeatedly until a condition or conditions are met -- so we can let a user try to log in as many times as they want until they get the password correct.

```
correct = "password"
password = ""

while password != correct:
     password = input("What is the password?: ")
```

We could also introduce other conditions, such as a password attempt limit.

More About `while` Loops

What's wrong with the following code?

```
while True:
    print("We're in a loop.")
print("We're out of the loop.")
```

It's important to be sure that your condition in a loop is met so that your program isn't stuck in the loop infinitely.

`break` is another way to escape a `while` loop. For example:

```
while True:
    username = input("Username: ")
```

While `continue` allows us to skip back to the top of the loop:

```
while True:
    username = input("Username: ")
    if (username != "David"):
        continue
    break
print "Hi David"
```

Logging In

Write a program password.py that asks for a name and a password and grants access if the username is ‘Mike’ and the password is ‘Awesome!’.

If the user types in a user other than ‘Mike’, the program will prompt for user again. If the user is correct, the program prompts for a password, and only grants access if that password is correct.

See sample output:

```
Who are you?: joe

Who are you?: Mike

What is your password, Mike?: Cool!

Who are you?: Mike

What is your password, Mike?: Awesome!

Access granted.
```

“Truthy” and “Falsy” Values

- Python considers some values equivalent to `True` or `False`
- `0, 0.0, ‘’, “”` are all considered “falsy” because if they are used as conditions, like `if 0,` the condition will evaluate to false.
- Strings that contain characters, and ints and floats that aren’t zero resolve to `True`
- When set to variable names, just using these values can make your code easier to read

```
sky_is_blue = True

if sky_is_blue:
```

Compared to

```
if sky_is_blue == True:
```

for loops & range()

What if you want to execute a block of code a certain number of times? You could use a `while` loop to do this, but there's an easier way, using a `for` loop with the `range()` function.

Here's a program written with a for loop?

```
print("My name is")

for i in range(5):

    print('Jimmy Five Times (' + str(i) + ')')
```

What does this program do? Figure 2-14 in the book may be a helpful reference.

Working with for loops: Gauss's Busy Work

Write a program sum.py that adds up all the numbers from 0 up to and including 100 using a `for` loop.

Print your final sum. It should be 5050.