Python

Python from the Terminal

These Python-related commands can be entered in bash. (That's the prompt with the `$`.) They won't work inside Python. (Python uses the >>> prompt.)

`python --version` - Prints out version information for Python, and will tell you if you're using Anaconda or the default Python. Useful for figuring out if Python has successfully installed on your computer, and whether it's accessible from the command line.

`python` - Typing Python by itself brings up the Python interpreter, also known as the Python shell or the Python REPL (which stands for "read–eval–print loop"). Lines you type after this will be sent directly to Python and evaluated. Any result will be printed back to you. A standard interaction might look like this:

```
Python 3.6.4 (default, Jan 15 2018, 23:11:13)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 2
4
```

`python <filename>` - Typing `python` and following it with a filename will feed that file into Python. This is how you run scripts from the command line.

Python Syntax and Commands

These commands and syntax are used in the Python interpreter (the >>>) or when writing a Python text file. They will not work in the bash shell (the `$` prompt).

Operators

Python accepts these operators:

- `+` - Plus. Add two things.
- `-` - Minus. Subtract one thing from another.
- `*` - Multiply. Multiply one thing by another.
- `/` - Divide. Divide one thing by another.
- `%` - Modulo or "mod." Divide and return the remainder..

Operators work on integers and floats, but can also be used to, for example, add strings together with plus or make them repeat with multiply.

Data Types

Data types are classifications of what a variable represents and what kind of operators can be performed on it. For example, you can subtract one integer from another, but you're not allowed to subtract one string from another.

The data types we use in this lesson are:

- *integer* - A whole number, a number without a decimal point. `1` is an integer.
- *float* - A floating point number, a number with a decimal point. `1.0` is a float.
- *string* - A series of arbitrary text characters, including letters, numbers, and symbols. Basically, some text. Strings are denoted by double quotes, single quotes, triple double quotes, or triple single quotes. `"Hello, world!"` is a string. `'''Hello, world!'''` is also a string.
- *list* - A list is a collection of items, which can be any other data type, including lists. Lists are denoted by a `[` at the beginning and a `]` at the end, with commas between items. `[1, 2, 3]` is a list. `["I", "have", 10, "mushrooms"]` is a list.
- *Boolean* - A binary choice between truthiness and falsiness. Boolean values are written with capital letters. `True` is a Boolean. `False` is a Boolean. The word "Boolean" is capitalized because it derives from a proper name.

Functions

Functions take some input and either do something or return a value or both. The input goes inside the parentheses.

print() - Print to the screen whatever is put in the parentheses.

type() - Return a value that represents the type of whatever is in the parentheses.

Variables

A variable is a name that is assigned to a value. Assignment looks like this:

```
x = 3
name = "Patrick"
name2 = "Ignatz"
grocery_list = ["tomato", "limes", "chocolate"]
```

Variable assignment uses a single `=`. The name given is on the left of the `=`, the value is on the right of the `=`.

Loops

A loop takes a list and moves through it one value at a time. For each of the values, it runs the code in the indented block. While running that code, the name after "for" can be used to refer to the current value.

```
flowers = ['rose', 'violet', 'buttercup']
for flower in flowers:
    print("My favorite flower is the", flower)

for <name_given_on_each_loop> in <list>:
    print("This is the current list item:", <name_given_on_each_loop>>
```

Remember that loops need a semicolon at the end of the first line and need to indent the code in the block below.

Comparison and Equality

Comparison operators check if an expression is true or false.

- `==` - Checks if a value is equal to another value. `3 == 3` returns `True`. `"love" == "love!"` returns `False`.
- `>` - Greater than. `10 > 2` returns `True`. `3 > 4` returns `False`.
- `<` - Less than. `3 < 4` returns `True`. `10 < 2` returns `False`.
- `!=` - Does not equal. `3 != 4` returns `True`. `10 != 10` returns `False`.

You can also use >= and <= (greater than or equal to and less than or equal to).

Conditionals

Conditionals check whether a value or comparison is true or false. If it's true, some code is run. If it's false, different code is run.

```
weather = "sunny"

if weather == "sunny":
    print("Bring your shades")
elif weather == "rainy":
    print("Bring your umbrella")
else:
    print("I don't know what you should bring! I'm just a little program...")
```

Like a loop, conditionals start a block with a colon at the end of the line.

Components of a conditional:

- *if statement* - Starts with `if` followed by a value that is either true or false. The block after the colon (`:`) is run if the statement is true.
- *elif statement* - Starts with `elif` followed by a value that is either true or false. Only used if the value after the initial `if` was false. If the value after `elif` is true, the code in the block underneath is run.
- *else statement* - A catchall. If the `if` statement was not true and none of the `elif` statements are true, run the code block under `else`.
- *case* - Each branch in the conditional, whether if, elif, or else, is called a **case**.

You can have multiple elif statements but only one if statement and only one else statement in a conditional. You don't need to have elif or else statements. If you don't have elif or else statements, if the if statment isn't true, nothing will happen.

Input

`input()` is a function that prints the string in parentheses, then waits for the user to type. After the user types and presses enter, the `input()` function uses the value the user entered within the program.

This two-line program asks the user a question and uses their answer in a response.

```
name = input("What's your name? ")
print(name + "? That's great. Not what I'd have chosen, but great.")
```

Importing Libraries

A **library** is a set of code, including functions and objects, that can be brought into your program and used.

Import a library like this:

```
import random
```

In this case, random is the name of our library, but we could also, for example, `import csv` if we wanted to work with CSVs.

To use a function from the random library, we use a special dot (`.`) syntax:

```
random_number = random.randint(0, 100)
print("My random number is", random_number)
```

Libraries Encountered in the DHRI Curriculum

Pandas

`pandas` is a package that can be used with Python to create navigable table structures called DataFrames, similar to Excel spreadsheets or databases.

A **DataFrame** is the highest structure of data in Pandas. DataFrames contain columns and rows, at which intersection's you will find your data points.

NLTK

See glossary for text analysis