Getting Started with Python

A workshop by Clara Turp, for McGill University Libraries

This workshop was built using this: https://librarycarpentry.org/lc-python-intro/

Please leave feedback:

Pre-Workshop: Downloading Anaconda

Installing Python Using Anaconda

https://librarycarpentry.org/lc-python-intro/setup.html

Python is great for general-purpose programming and is a popular language for scientific computing as well. Installing all of the packages required for this lessons individually can be a bit difficult, however, so we recommend the all-in-one installer Anaconda.

Regardless of how you choose to install it, please make sure you install Python version 3.x (e.g., Python 3.6 version). Also, please set up your Python environment at least a day in advance of the workshop. If you encounter problems with the installation procedure, ask your workshop organizers via e-mail for assistance so you are ready to go as soon as the workshop begins.

Windows - Video tutorial

1. Open anaconda.com/download with your web browser.
2. Download the Python 3 installer for Windows.
3. Double-click the executable and install Python 3 using *MOST* of the default settings. The only exception is to check the **Make Anaconda the default Python** option.

macOS - Video tutorial

1. Open anaconda.com/download with your web browser.
2. Download the Python 3 installer for macOS.
3. I recommend choosing the Graphical Installer
4. Install Python 3 using all of the defaults for installation.

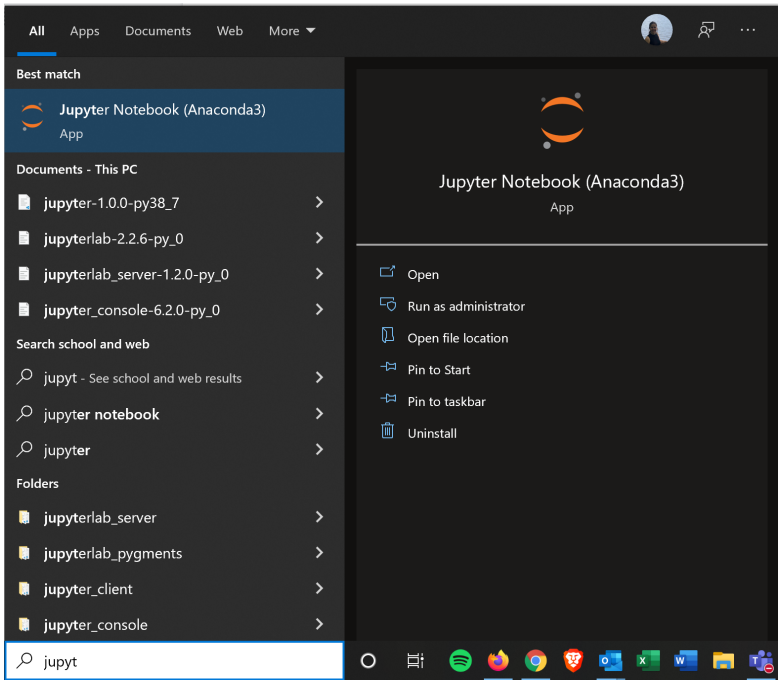
Getting Started

https://librarycarpentry.org/lc-python-intro/01-getting-started/index.html

Use Jupyter

- The Anaconda package manager is an automated way to install Jupyter Notebooks..

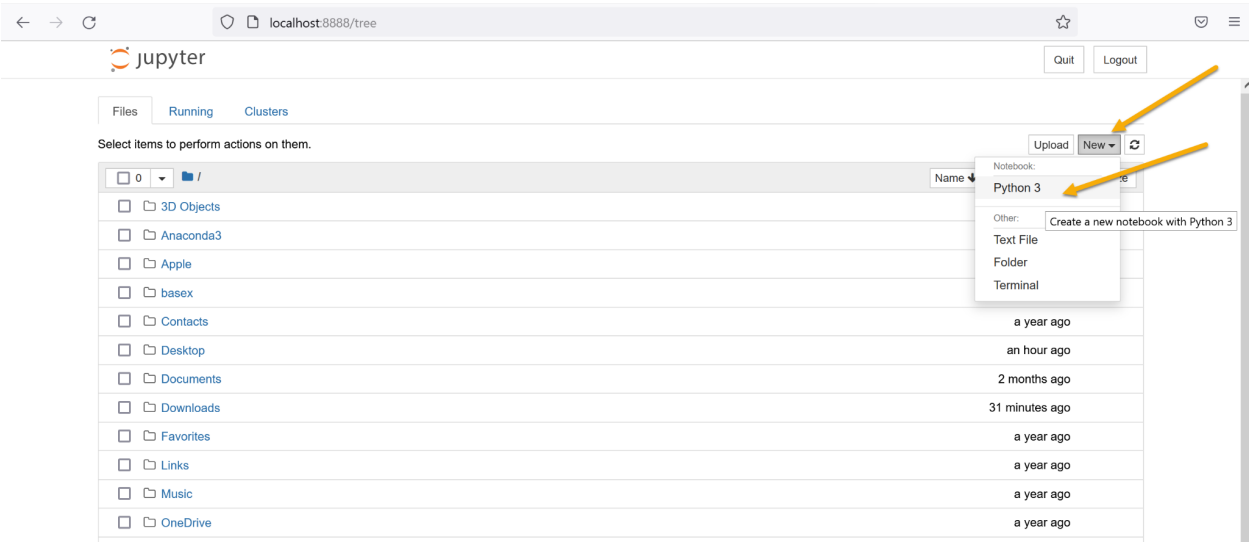
Once you have installed Python, you can search your computer for the Jupyter app.



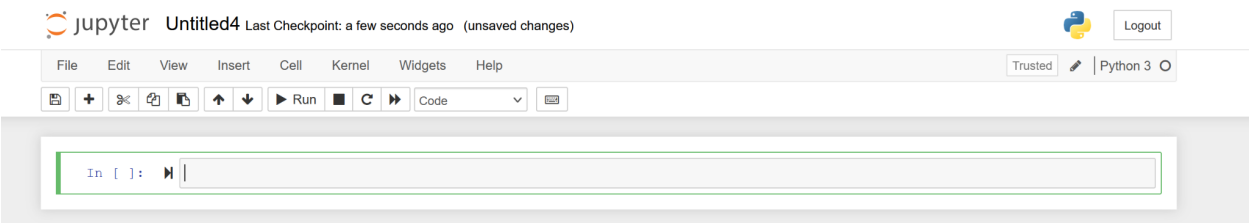
- This will start Jupyter Notebooks in your web browser. You will also see the application run in a terminal.

The advantages of Jupyter is that you can create a notebook with code that you can run and save the results of your code directly under it. You can also add notes.

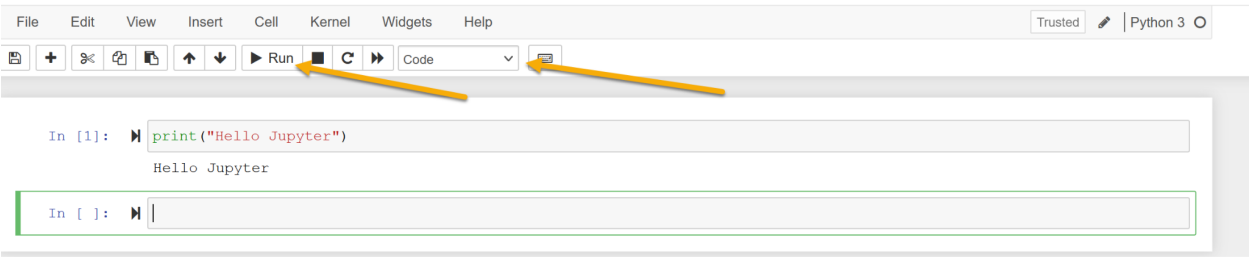
In Jupyter, click on **New** and then on **Python 3 Notebook**



A new notebook will be created in a new tab.

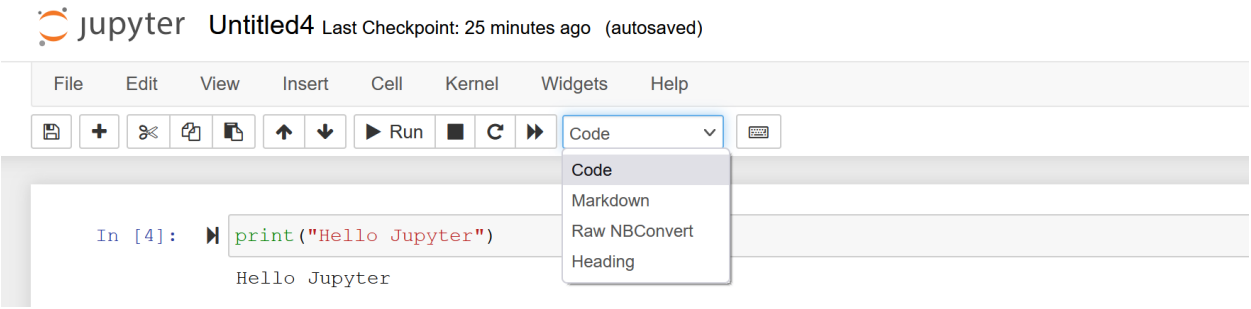


Try to type some Python code in the top box, next to the arrow, such as **print("Hello Jupyter")**. You can then run your code by clicking on the **Run button**, or with a keyboard shortcut **Shift-Enter.** You will then see the code you wrote and the result. A new cell will appear underneath. If you made a mistake, you can still edit the cell above.



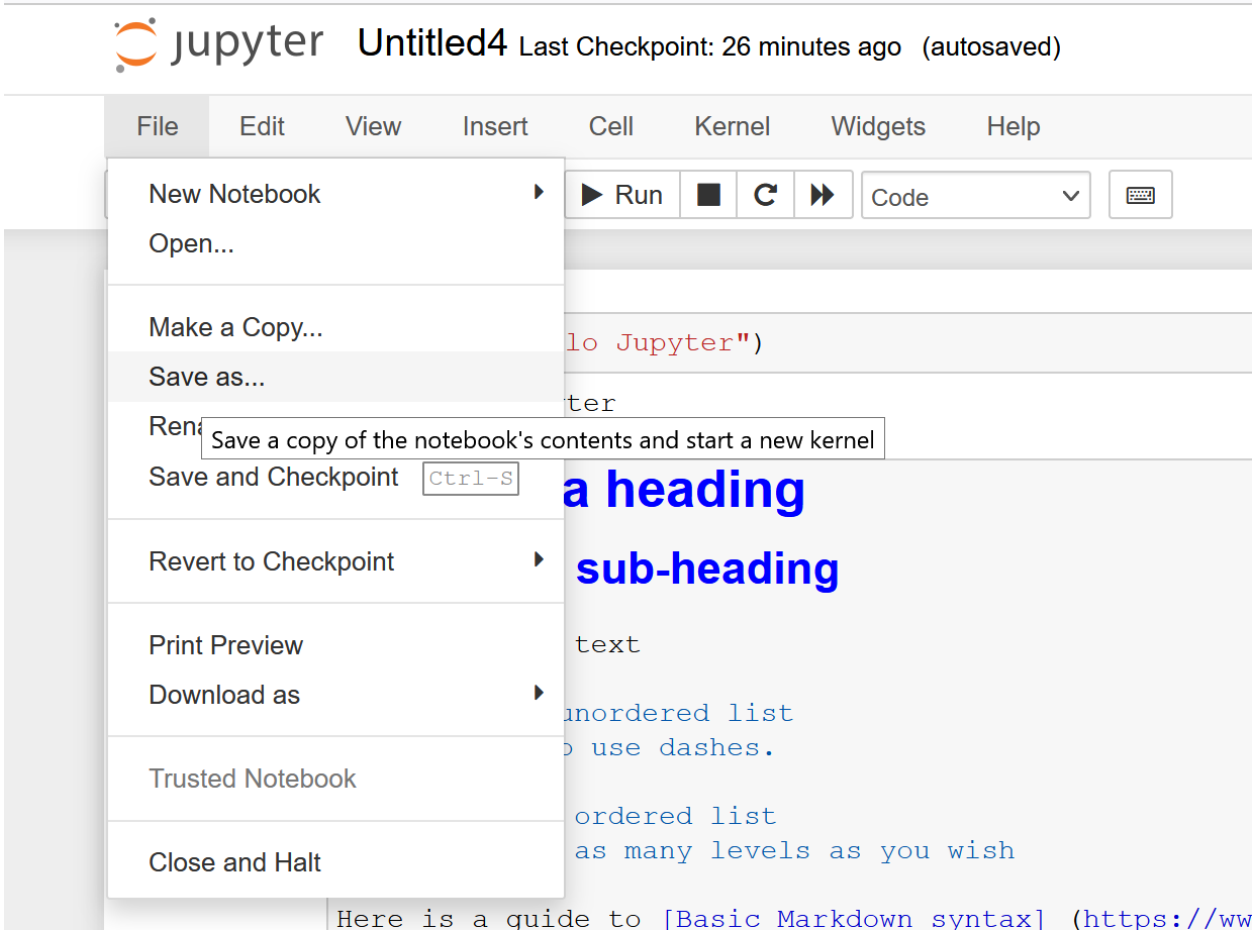
You can also add new cells by clicking on the **+** sign, which is to insert a cell below. You can also select Insert and choose whether you want to insert a cell above or below your current cell.

The great part of Jupyter Notebooks is that it combines different types of cells, you can add **code** cells or **Markdown** cells. This means you can include all your workshop notes directly with the code.

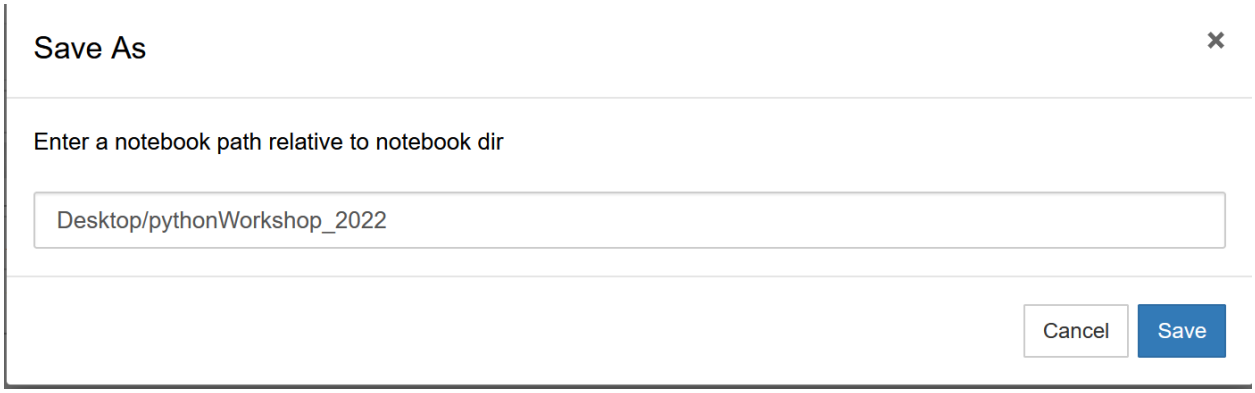


Saving the code

To save the code, press ‘file’ and then ‘save as’. This saves your notebook (using the ipynb extension). This saves the notebook at your current directory in the Jupyter home page.



If I wanted to save it on my **Desktop**, I would have to type Desktop/PythonWorkshop_2022. This is called a relative path.



Use comments to add documentation to programs.

You can add comments to document your script, this works in Jupyter, but also wherever you are writing Python. Usually comments are used to describe what the script is doing. General documentation that might help your future-you.

- For one-line comments, use hash (#). The commented line will be ignored when you run a script.
- For multi-line comments, use three quotation marks (“””).
- Use comments to debug

```
#This is a comment

"""
This is a long comment

Usually when you have a multi-line comment

"""
```

Variables and Assignment

https://librarycarpentry.org/lc-python-intro/02-variables/index.html

Use variables to store values, and some data types.

- Variables are names for values.
- In Python the = symbol assigns the value on the right to the name on the left.
- The variable is created when a value is assigned to it.

Some data types:

- Integers (numbers) - don't use quotation marks.
- Strings (words) - use quotation marks
- If we type a number between quotation marks, such as Title = "1984" , what is it?

- Here, Python assigns an age to a variable age and a name in quotation marks to a variable first_name.
- Can you see the different colors for the different data types?

In [ ]:
```
age = 35              #Integers (numbers)
first_name = "Clara" #Strings (words), use quotation marks.
title = "1984"        #What would this be?
```

- You can see here that I added comments with the hash (#). You can add comments to document your script, this works in Jupyter, but also wherever you are writing Python. Usually comments are used to describe what the script is doing. General documentation that might help your future-you.

- Now if we called a variable, it would represent the value stored in it. So print(age) will print out the value assigned to the variable age, which is 35. To call the variable, you type the variable name without any quotation marks.

In [1]:
```
age = 35              #Integers (numbers)
first_name = "Clara" #Strings (words), use quotation marks.
title = "1984"        #What would this be?

print(age)
```

```
35
```

- Variable names:
    - Should have a meaning
    - cannot start with a digit
    - cannot contain spaces, quotation marks, or other punctuation
    - Use underscore or CamelCase (typically used to separate words in long variable names)
        - Underscores at the start like __alistairs_real_age have a special meaning so don't use it.

Python is case-sensitive.

- Python thinks that upper- and lower-case letters are different, so Name and name are different variables.
- There are conventions for using upper-case letters at the start of variable names so we will use lower-case letters for now.

Use print to display values.

- As we saw, Python has a built-in function called print that prints things as text.
- Call the function (i.e., tell Python to run it) by using its name.
- Provide values to the function (i.e., the things to print) in parentheses.
- To add a string to the printout, wrap the string in quotations.
- The values passed to the function are called 'arguments'
    - So Print is a function (we will explore this later)
    - Print expects at least one argument
    - The arguments can be a variable, an integer, a string.
    - The arguments will be printed to the console.

If we want to print Clara is 35 years old, we could do: print("Clara is 35 years old")

Or we could use the variables we defined earlier. Please note that Python remembers the variables, even if they were created earlier.

Variables need to be created before they are used.

```
In [2]: # Print is a built-in function that prints the function's argument as text.
        print("Clara is 35 years old")
        print(first_name, "is", age, "years old")

        Clara is 35 years old
        Clara is 35 years old
```

Here is what happens once we run the script:

- print automatically puts a single space between items to separate them; items (arguments) are separated by a comma.
- And wraps around to a new line at the end.

Variables can be used in calculations.

- We can use variables in calculations just as if they were values.
    - Remember, we assigned 35 to age.
- There are multiple ways to add 3 to Clara's age.
    - The first one (age + 3) prints out 38, however that value will not be saved.
    - The second one (age = age + 3) replaces the value of age with 38.
    - The third one (age_plus_three = age + 3) assigns the value 38 to a new variable. You can see here that it prints 41, because we changed the value of age.

The three different options for calculating age + 3

```
In [2]: age = 35

        print(age + 3)
```

```
38
```

```
In [3]: age = 35
        age = age + 3

        print(age)
```

```
38
```

```
In [5]: age = 35
        age_plus_three = age + 3

        print(age_plus_three, age)
```

```
38 35
```

Some built-in functions and methods

https://librarycarpentry.org/lc-python-intro/04-built-in/index.html

https://www.w3schools.com/python/python_ref_string.asp

Built-in functions

Functions are central to programming. You can write functions or use some of the built-in functions provided. Functions allow you to take an action (algorithm) on an element you link it to. Functions are powerful because you can use them repeatedly.

A function may take zero or more arguments.

- We have seen some functions already — now let's take a closer look.
- An *argument* is a value passed into a function.
- Any arguments you want to pass into a function must go into the ()
    - print("I am an argument and must go here.")
- You must always use parentheses, because this is how Python knows you are calling a function.
    - You leave them empty if you don't want or need to pass any arguments in.
- print takes zero or more.

Every function returns something.

- Every function call produces some result.
- If the function doesn't have a useful result to return, it usually returns the special value None.

Use the built-in function len to find the length of a string.

- Len() is a built-in function. It returns the length of an objectL
    - How many characters are in a string
    - How many items are in a list
    - Length function doesn't work on numbers.
- For now, let's worry about the length of a string.
- The function takes one argument in the parentheses.

```
element = "Helium"
print(len(element))
```

```
6
```

- Nested functions are evaluated from the inside out, just like in mathematics.
    - Finding out the length of the variable element, before you print.

Use string methods - upper() and lower() - to transform a string

- A method is similar to functions, but they are tied to an object (i.e. a string)
    - Use object_name.method_name to call methods.
- These methods take a string and return the same string with all uppercase or lowercase.

```
element = "helium"
upper_element = element.upper()
print(upper_element)
```

```
HELIUM
```

Lower exercise:

- Create a variable “python” and assign it the phrase “Hello World, I AM learning Python.”
- Use the lower method to transform the phrase to only lowercase.
- Print the result to the console.

    python = “Hello World, I AM learning Python”
    lower_python = python.lower()
    print(lower_python)

Activity

We would like to create a script that reads files from a folder and returns ”Match found” with the filename when a match is found for a specific word.
We will build this script slowly during this workshop. The first thing is to add files to a new folder you created in the folder for this workshop.

Let’s create a variable called numberOfMatches and assign it the value 0.

```
numberOfMatches = 0
```

Lists

https://librarycarpentry.org/lc-python-intro/11-lists/index.html

A list stores many values in a single structure.

- Scenario: You want to keep track of many temperatures.
- Doing calculations with a hundred variables called temperature_001, temperature_002, etc., would be at least as slow as doing them by hand.
- You can create a list by assigning brackets ([]) to a list name. You can create it with values or without.
    - Use [] on its own to represent a list that doesn’t contain any values.
        - “The zero of lists.”
- Use a *list* to store many values together.
    - Contained within square brackets [...].
    - Values separated by commas ,.
- Use len to find out how many values are in a list.

```
temperatures = []
print(temperatures)
```

```
[]
```

```
temperatures = [17.3, 17.5, 17.7, 17.5, 17.6]
print(temperatures)
print("length of temperatures list", len(temperatures))
```

```
[17.3, 17.5, 17.7, 17.5, 17.6]
length of temperatures list 5
```

Use an item's index to fetch it from a list.

- The count starts at 0.
- We can use this on strings too.

```
print(temperatures[0])
print(temperatures[4])
```

```
17.3
17.6
```

Exercise

- Try to return the last letter of the string using len(). Note: If Python starts counting from zero, and len returns the number of characters in a string, what index expression will get the last character in the string name?
    - name[len(name) - 1]
- Try to return the last element of the temperatures list using len()

Appending items to a list lengthens it.

- Use list_name.append to add items to the end of a list.

```
temperatures = [17.3, 17.5, 17.7, 17.5, 17.6]

temperatures.append(19.7)
print(temperatures)
```

```
[17.3, 17.5, 17.7, 17.5, 17.6, 19.7]
```

- append is a *method* of lists.

Conditionals

https://librarycarpentry.org/lc-python-intro/17-conditionals/index.html

Use if statements to control whether or not a block of code is executed.

- An if statement (more properly called a *conditional* statement) controls whether some block of code is executed or not.
- The idea is:
    - If x then:
        - Do a
    - If y then:
        - Do b
    - A is only executed if the conditions of x are met.
- Structure is similar to a for statement:
    - If variable conditions :
        - Block of code
        - This block of code is indented (usually by 4 spaces)
    - The conditions often use <, >, ==, !=

```
#mass = 3.54
mass = 2.97
if mass > 3.0:
    print(mass, "is larger")
if mass < 3.0:
    print(mass, "is smaller")
```

```
2.97 is smaller
```

- Indentation is always meaningful in Python. Expected indented block missing or unexpected indent will return an error.

```
firstName = "John"
 lastName = "Smith"
```

```
  File "<ipython-input-8-da09491f9801>", line 2
    lastName = "Smith"
    ^
IndentationError: unexpected indent
```

Activity:

Create an if statement that evaluates whether the numberOfMatches variable is greater than 0, meaning that there would be at least one match.

```
if numberOfMatches > 0:
    print("match found")
```

Use elif to specify additional tests and else to execute a block when the conditions are not true (the catch all).

- May want to provide several alternative choices, each with its own test.
- Use elif (short for “else if”) and a condition to specify these.
- Always associated with an if.
- Once the condition is met, the code stops. That is why 9.7 isn’t printed twice, once with is HUGE and once with is larger.
- Must come before the else (which is the “catch all”).

```
mass = 4.9
if mass > 9.0:
    print(mass, "is huge")
elif mass > 3.0:
    print(mass, "is larger")
else:
    print(mass, "is smaller")
```

```
4.9 is larger
```

Conditions are tested once, in order.

- Python steps through the branches of the conditional in order, testing each in turn.
- So ordering matters.
- How can we fix this error?

```python
grade = 85

if grade >= 70:
    print("Grade is C")
elif grade >= 80:
    print("Grade is B")
elif grade >= 90:
    print("Grade is A")
```

```
Grade is C
```

For Loops

https://librarycarpentry.org/lc-python-intro/12-for-loops/index.html

A *for loop* executes commands once for each value in a collection.

- Doing calculations on the values in a list one by one is as painful as working with temperature_001, temperature_002, etc.
- Let's say you want to print out every temperature recorded. You could do it this way:

```
temperature_01 = 17.5
temperature_02 = 15.9
temperature_03 = 19.4
temperature_04 = 21.1

print(temperature_01)
print(temperature_02)
print(temperature_03)
print(temperature_04)
```

- You can also use a for loop to do this systematically.
- A *for loop* tells Python to execute some statements once for each value in a list, a character string, or some other collection.
- "for each thing in this group, do these operations"

```
for temperature in [17.5, 15.9, 19.4, 21.2]:
    print(temperature)
```

- You can also do this with a list assigned to a variable:

```
temperatures = [17.5, 15.9, 19.4, 21.2]

for current_temperature in temperatures:
    print(current_temperature)
```

```
17.5
15.9
19.4
21.2
```

- The output of all these should be the same.:

The first line of the for loop must end with a colon, and the body must be indented.

- Just like conditionals, the indentation is essential.
- The colon at the end of the first line signals the start of a *block* of statements.
- Python uses indentation rather than {} or begin/end to show *nesting*.

A for loop is made up of a collection, a loop variable, and a body.

```
for number in [2, 3, 5]:
    print(number)
```

- The collection, [2, 3, 5], is what the loop is being run on.
- The body, print(number), specifies what to do for each value in the collection. The body can contain many statements.
- The loop variable, number, is what changes for each *iteration* of the loop.
    - The “current thing”.

```
for number in [2,3,5]: # number is the loop variable. [2,3,5] is the collection, what the loop is running on.
    print(number) # The body specifies what to do for each value in the collection.
```

Use range to iterate over a sequence of numbers.

- The built-in function range produces a sequence of numbers.
    - *Not* a list: the numbers are produced on demand to make looping over large ranges more efficient.
- range(N) is the numbers 0…N-1

```
for number in range(0,3):
    print(number)
```

```
0
1
2
```

The Accumulator pattern turns many values into one.

- A common pattern in programs is to:
    1. Initialize an *accumulator* variable to zero, the empty string, or the empty list.

2. Update the variable with values from a collection.

```
# sum of the first 10 integers.

total = 0 # Accumulator variable
for number in range(10):
    total = total + (number + 1) # Remember, the range will start at 0.

print(total)
```

```
55
```

- Read total = total + (number + 1) as:
    - Add 1 to the current value of the loop variable number.
    - Add that to the current value of the accumulator variable total.
    - Assign that to total, replacing the current value.
- We have to add number + 1 because range produces 0..9, not 1..10.

Activity

- You can use loops and conditionals to “evolve” the values of variables. In the activity notebook, create a for loop that reads all lines in the reader variable. Then make the line lower case. With each line, create a first if statement that looks for the word sleep in line.
- Then If the word is found, add 1 to the variable numberOfMatches, using the accumulator pattern.

```
for line in reader:
    line = line.lower()
    if "sleep" in line:
        numberOfMatches = numberOfMatches + 1
```

Using libraries

https://librarycarpentry.org/lc-python-intro/06-libraries/index.html

Most of the power of a programming language is in its (software) libraries.

- A *(software) library* is a collection of files (called *modules*) that contains functions for use by other programs.
    - Library's contents are supposed to be related, but there's no way to enforce that.
- The Python standard library is an extensive suite of modules that comes with Python itself.
- Many popular libraries come with Anaconda.
- Libraries allow you to not reinvent the wheel.

A program must import a library module before using it.

- Use import to load a library module into a program's memory.
- Then refer to things from the module as module_name.thing_name.
    - Python uses . to mean "part of".
- For the activity, we will import a library: os
    - Os: Module to interact with the operating system.
    - For example, read or write a file, manipulate path

```
# import a library
import os
```

```
currentDir = os.getcwd()
print(currentDir)
```

```
C:\Users\cturp1\Desktop\PythonWorkshop
```

Activity

Let's walk through the rest of the script, using the function we already wrote, the variable with created, and the if statement.

```
# import a library
import os

## Script:

currentDir = os.getcwd()
filesDir = currentDir + "\\FairyTales\\"

#Python method listdir() returns a list of files / folders in the directory given.
for filename in os.listdir(filesDir):

    numberOfMatches = 0
    currentFile = filesDir + filename

    if filename != ".ipynb_checkpoints":
        myFile = open(currentFile, "r", encoding = "utf-8", errors = "ignore")
        reader = myFile.readlines()
        for line in reader:
            line = line.lower()
            if 'sleep' in line:
                numberOfMatches = numberOfMatches + 1

    if numberOfMatches > 0:
        print("match found", numberOfMatches, "times in:", filename)
```