Str a = “what is your name”

Strip() = will remove extra space ex: print(this.strip())

Int a = 21

List = []

Sum() will give total of list

Len() will give length of list

.append() to add item to list

.remove() to remove any req item

Dictionary = {key : Value}

.count() to count items

.uppper() for upper case

.lower() for lower case

Sorted() to sort the list in alphabaticle order

dir(list) To check what methods are available we use

dir(\_\_buildins\_\_) will give complete list of functions available

.replace(): to replace items, print(“todai”.replace(“i”,”y”))

help(len) will give info about function

(args) is used to take multiple arguments in function

**(\*\*kwargs) is used to take multiple key word arguments in function and it will return dictionary**

* You can get a list of **attributes** of a data type has using:

1. dir(str)
2. dir(list)
3. dir(dict)

* You can get a list of Python **builtin functions** using:

1. dir(\_\_builtins\_\_)

* You can get the **documentation** of a Python data type using:

1. help(str)
2. help(str.replace)
3. help(dict.values)

5/17400 panorama center 1 2nd 232

8/25200

10/29600

Python is mainly used for automation purposes, web apps, and data science. Many big companies, like Instagram, Facebook, and Amazon, use Python in different parts of their products. For example, Facebook uses Python to process images.

**Tip: Converting Between Datatypes**

Sometimes you might need to convert between different data types in Python for one reason or another. That is very easy to do:

**From tuple to list:**

1. >>> cool\_tuple = (1, 2, 3)
2. >>> cool\_list = list(cool\_tuple)
3. >>> cool\_list
4. [1, 2, 3]

**From list to tuple:**

1. >>> cool\_list = [1, 2, 3]
2. >>> cool\_tuple = tuple(cool\_list)
3. >>> cool\_tuple
4. (1, 2, 3)

**From string to list:**

1. >>> cool\_string = "Hello"
2. >>> cool\_list = list(cool\_string)
3. >>> cool\_list
4. ['H', 'e', 'l', 'l', 'o']

**From list to string:**

1. >>> cool\_list = ['H', 'e', 'l', 'l', 'o']
2. >>> cool\_string = str.join("", cool\_list)
3. >>> cool\_string
4. 'Hello'

As can be seen above, converting a list into a string is more complex. Here str() is not sufficient. We need str.join(). Try running the code above again, but this time using str.join("---", cool\_list) in the second line. You will understand how str.join() works.

**Cheatsheet: Functions and Conditionals**

In this section, you learned to:

* Define **functions**:

1. def cube\_volume(a):
2. return a \* a \* a

* Write **if-else** **conditionals**:

1. message = "hello there"
3. if "hello" in message:
4. print("hi")
5. else:
6. print("I don't understand")

* Write **if-elif-else conditionals:**

1. message = "hello there"
3. if "hello" in message:
4. print("hi")
5. elif "hi" in message:
6. print("hi")
7. elif "hey" in message:
8. print("hi")
9. else:
10. print("I don't understand")

* Use the and operator to check if **both conditions** are True at the same time:

1. x = 1
2. y = 1
4. if x == 1 and y==1:
5. print("Yes")
6. else:
7. print("No")

* Use the or operator to check if **at least one condition** is True:

1. x = 1
2. y = 2
4. if x == 1 or y==2:
5. print("Yes")
6. else:
7. print("No")

* Check if a value is of a particular **type** with **isinstance**:

1. isinstance("abc", str)
2. isinstance([1, 2, 3], list)

or directly:

1. type("abc") == str
2. type([1, 2, 3]) == lst

**Cheatsheet: Processing User Input**

In this section, you learned that:

* A Python program can get **user input** via the input function:
* The **input** **function** halts the execution of the program and gets text input from the user**:**

1. name = input("Enter your name: ")

* The input function converts any **input to a string**, but you can convert it back to int or float:

1. experience\_months = input("Enter your experience in months: ")
2. experience\_years = int(experience\_months) / 12

* You can also **format strings** with:

1. name = "Sim"
2. experience\_years = 1.5
3. print("Hi {}, you have {} years of experience".format(name, experience\_years))

Output: Hi Sim, you have 1.5 years of experience.

**For Loop Over a Function**

A for loop can also be used to execute a function multiple times. For example, below we are executing celsius\_to\_kelvin three times since there are three items in the iterating list:

1. def celsius\_to\_kelvin(cels):
2. return cels + 273.15
4. for temperature in [9.1, 8.8, -270.15]:
5. print(celsius\_to\_kelvin(temperature))

The output of that would be:

282.25  
281.95  
3.0

So, in the first iteration celsius\_to\_kelvin(9.1) was executed, in the second celsius\_to\_kelvin(8.8) and in the third celsius\_to\_kelvin(-270.15).

**Dictionary Loop and String Formatting**

Here is an example that combines a dictionary loop with string formatting. The loop iterates over the dictionary and it generates and prints out a string in each iteration:

1. phone\_numbers = {"John": "+37682929928", "Marry": "+423998200919"}
3. for pair in phone\_numbers.items():
4. print(f"{pair[0]} has as phone number {pair[1]}")

And here is a better way to achieve the same results by iterating over keys and values:

1. phone\_numbers = {"John": "+37682929928", "Marry": "+423998200919"}
3. for key, value in phone\_numbers.items():
4. print(f"{key} has as phone number {value}")

In both cases, the output is:

John has as phone number +37682929928

Marry has as phone number +423998200919

**Cheatsheet: Loops**

In this section, you learned the following:

* A**for-loop** is useful to repeatedly execute a block of code.
* You can create a **for-loop** like so:

1. for letter in 'abc':
2. print(letter.upper())

Output:

A  
B  
C

As you can see, the for-loop repeatedly converted all the items of 'abc' to uppercase.

* The name after for (e.g. letter) is just a variable name
* You can loop over **dictionary keys**as follows:

1. phone\_numbers = {"John Smith":"+37682929928","Marry Simpons":"+423998200919"}
2. for value in phone\_numbers.keys():
3. print(value)

Output:

John Smith  
Marry Simpsons

* You can loop over **dictionary values**:

1. phone\_numbers = {"John Smith":"+37682929928","Marry Simpons":"+423998200919"}
2. for value in phone\_numbers.values():
3. print(value)

Output:

+37682929928  
+423998200919

* You can loop over **dictionary items**:
  1. phone\_numbers = {"John Smith":"+37682929928","Marry Simpons":"+423998200919"}
  2. for key, value in phone\_numbers.items():
  3. print(key, value)

Output:

* 1. John Smith +37682929928
  2. Marry Simpons +423998200919
* We also have **while-loops**.The code under a while-loop will run as long as the while-loop condition is true:
  1. while datetime.datetime.now() < datetime.datetime(2090, 8, 20, 19, 30, 20):
  2. print("It's not yet 19:30:20 of 2090.8.20")

The loop above will print out the string inside print() over and over again until the 20th of August, 2090.

**Cheatsheet: List Comprehensions**

In this section, you learned that:

* A list comprehension is an expression that creates a list by iterating over another container.
* A **basic**list comprehension:
  1. [i\*2 for i in [1, 5, 10]]

Output: [2, 10, 20]

* List comprehension with **if** condition:
  1. [i\*2 for i in [1, -2, 10] if i>0]

Output: [2, 20]

* List comprehension with an **if** **and** **else** condition:
  1. [i\*2 if i>0 else 0 for i in [1, -2, 10]]

Output: [2, 0, 20]

**Cheatsheet: More on Functions**

In this section, you learned that:

* Functions can have more than one **parameter**:

1. def volume(a, b, c):
2. return a \* b \* c

* Functions can have **default** parameters (e.g. coefficient):

1. def converter(feet, coefficient = 3.2808):
2. meters = feet / coefficient
3. return meters
5. print(converter(10))

Output: 3.0480370641306997

Arguments can be passed as **non-keyword** (positional) arguments (e.g. a) or **keyword** arguments (e.g. b=2 and c=10):

1. def volume(a, b, c):
2. return a \* b \* c
4. print(volume(1, b=2, c=10))

* An **\*args**parameter allows the  function to be called with an arbitrary number of non-keyword arguments:

1. def find\_max(\*args):
2. return max(args)
3. print(find\_max(3, 99, 1001, 2, 8))

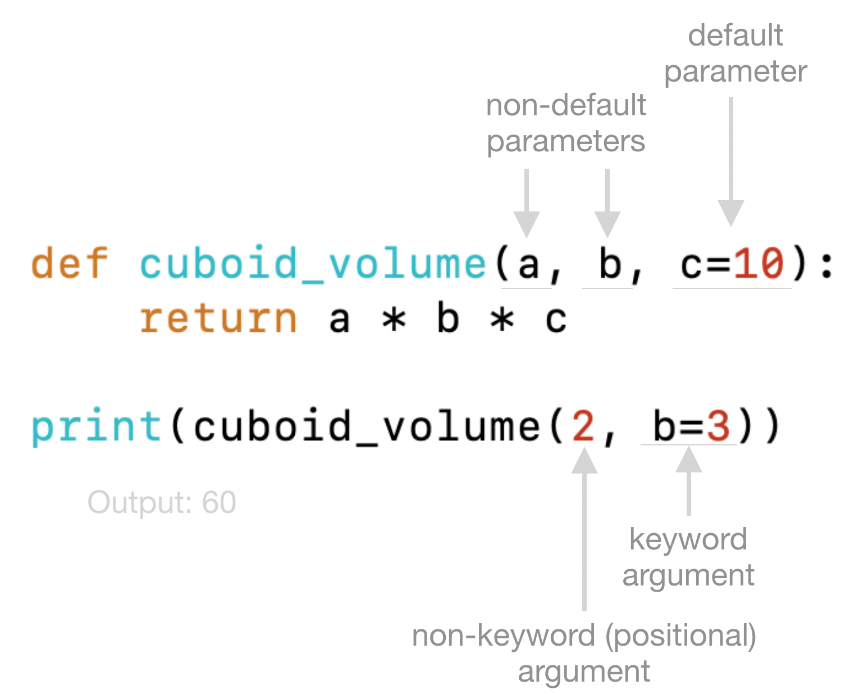
Output: 1001

* A **\*\*kwargs** parameter allows the function to be called with an arbitrary number of keyword arguments:

1. def find\_winner(\*\*kwargs):
2. return max(kwargs, key = kwargs.get)
4. print(find\_winner(Andy = 17, Marry = 19, Sim = 45, Kae = 34))

Output: Sim

* Here's a summary of function elements:



**Cheatsheet: File Processing**

In this section, you learned that:

* You can **read** an existing file with Python:

1. with open("file.txt") as file:
2. content = file.read()

* You can **create** a new file with Python and **write** some text on it:

1. with open("file.txt", "w") as file:
2. content = file.write("Sample text")

* You can **append** text to an existing file without overwriting it:

1. with open("file.txt", "a") as file:
2. content = file.write("More sample text")

* You can both **append and read** a file with:

1. with open("file.txt", "a+") as file:
2. content = file.write("Even more sample text")
3. file.seek(0)
4. content = file.read()

**Cheatsheet: Imported Modules**

In this section, you learned that:

* **Builtin objects** are all objects that are written inside the Python interpreter in C language.
* **Builtin modules** contain builtins objects.
* Some builtin objects are not immediately available in the global namespace. They are parts of a builtin module. To use those objects the module needs to be **imported** first. E.g.:
  1. import time
  2. time.sleep(5)
* **A list of all builtin modules** can be printed out with:
  1. import sys
  2. sys.builtin\_module\_names
* **Standard libraries** is a jargon that includes both builtin modules written in C and also modules written in Python.
* **Standard libraries** written in Python reside in the Python installation directory as *.py* files. You can find their directory path with sys.prefix.
* **Packages** are a collection of *.py* modules.
* **Third-party libraries** are packages or modules written by third-party persons (not the Python core development team).
* Third-party libraries can be **installed** from the terminal/command line:

Windows:

pip install pandas or use python -m pip install pandas if that doesn't work.

* Mac and Linux:

pip3 install pandas or use python3 -m pip install pandas if that doesn't work.

**Installing pandas**

Please make sure you have pandas installed. You can install it with pip from your computer or Atom/VS Code terminal/cmd just like you have installed other third-party packages. Please execute one of the commands below to do the installation depending on what version of Python you are using:

pip3.10 install pandas

or

pip3.9 install pandas

or

pip3.8 install pandas

etc.

Also, in the next lecture, we will use an enhanced Python interactive shell called IPython.

IPython is just like the standard shell you get when you run Python, but IPython provides better printing for large text. This ability makes IPython suitable for data analysis because the program prints data in a well-structured format. You can install IPython with pip:

pip3.10 install ipython

or

pip3.9 install ipython

or

pip3.8 install ipython

**Installing Jupyter**

In the next videos, we will use Jupyter Notebook is a third-party library. Please install Jupyter Notebook by executing one of the commands below:

**Installing Jupyter**

pip3.10 install jupyter

or

pip3.9 install jupyter

or

pip3.8 install jupyter

*Please change the number after pip to reflect the version of Python you are using.*

**Starting Jupyter**

After a successful installation you can start Jupyter Notebook by running the command below from your terminal:

jupyter notebook

If the above command does not work try the following command:

py -3.9 -m jupyter notebook (for Windows users)

python3.9 -m jupyter notebook  (for Mac and Linux users)

If it works, you will see Jupyter Notebook opened up in your default internet browser.

**Jupyter in the cloud**

If you do not want to install Jupyter or you cannot install it, you can use Jupyter in the cloud. Google offers a ready-to-use Jupyter notebook here: <https://colab.research.google.com/#create=true>

**Exercise: Loading JSON Files**

In  the previous lecture, you learned that you can load a CSV file with this code:

1. import pandas
2. df1 = pandas.read\_csv("supermarkets.csv")

Try loading the supermarkets.json file for this exercise using read\_json instead of read\_csv.

*The supermarkets.json file can be found inside the supermarkets.zip file attached in the previous lecture.*

**Solution: Loading JSON Files**

The code for loading the supermarkets.json file in Python with pandas would be this:

1. import pandas
2. df2 = pandas.read\_json("supermarkets.json")

The df2 dataframe should contain this data:



**Note on Loading Excel Files**

In the next lecture, you will learn how to load Excel files in Python with *pandas*. For this, you need *pandas* (which you have already installed) and also two other dependencies that *pandas* needs for opening Excel files. You can install them with *pip*:

pip3.9 install openpyxl (needed to load Excel *.xlsx* files)

pip3.9 install xlrd (needed to load Excel old *.xls* files)

pip install geopy for data frames

1. from geopy.geocoders import ArcGIS
2. nom = ArcGIS()

**Note**

We are going to use Nominatim() in the next video. Nominatim() currently has a bug. To fix this problem, whenever you see these lines in the next video:

1. from geopy.geocoders import Nominatim
2. nom = Nominatim()

change them to these

1. from geopy.geocoders import ArcGIS
2. nom = ArcGIS()

The rest of the code remains the same.

**Installing OpenCV**

In the next lecture, and in Section 17, we will use the OpenCV image processing library. Let us first make sure you have installed the OpenCV library. OpenCV is also referred to as cv2 in Python.

**How to Install OpenCV**

To install OpenCV for Python 3.9 on Mac or Linux, execute the following in the terminal:

python3.9 -m pip install opencv-python

To install OpenCV for Python 3.9 on Windows, execute the following in the terminal:

py -3.9 -m pip install opencv-python

**Note: The above commands work for Python 3.9. You may need to replace the 3.9 part from the commands with the number of the Python version you are using in your system. For example, you may need to type**python3.10**instead of**python3.9.

Once the installation completes, open a Python session and try:

import cv2

If you get no errors, you installed OpenCV successfully. If you get an error, see the FAQs below:

**FAQs**

**1. My OpenCV installation didn't work on Windows**

Solution:

1. Uninstall OpenCV with:

py -3.9 -m pip uninstall opencv-python

2. Download a wheel (.whl) file from [this link](http://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv) and install the file with pip. Make sure you download the correct file for your Windows and your Python versions. For example, for Python 3.6 on Windows 64-bit, you would do this:

py -3.9 -m pip install opencv\_python‑3.2.0‑cp39‑cp39m‑win\_amd64.whl

3. Try to import cv2 in Python again. If there's still an error, type the following again in the command line:

py -3.9 -m pip install opencv-python   
4. Try importing cv2 again. It should work at this point.

**2. My OpenCV installation didn't work on Mac**

Solution:

If python3.9 -m pip install opencv-python here are alternative steps to install OpenCV:

1. Install *brew*.

To install *brew,*open your terminal, and execute the following:

/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

2. OpenCV depends on GTK+, so install that dependency first with brew (always from the terminal):

brew install gtk+

3. Install OpenCV with brew:

brew install opencv

4. Open Python and try to import OpenCV with:

import cv2

If you get no errors, you installed OpenCV successfully.

**3. My OpenCV installation didn't work on Linux**

1. Open your terminal and execute the following commands, one by one:

1. sudo apt-get install libqt4-dev
2. cmake -D WITH\_QT=ON ..
3. make
4. sudo make install

2. 2. If the above commands don't work, execute this:

1. sudo apt-get install libopencv-\*

3. Then, install OpenCV with pip:

python3.9 -m pip install opencv-python

4. Import cv2 in Python. If you get no errors, you installed OpenCV successfully.

Resources for this lecture

**Note**

In the next lecture, I use this in the code:

tiles = "Mapbox Bright"

Please use this instead:

tiles = "Stamen Terrain"

Mapbox Bright and Stamen Terrain are both types of base maps, but Mapbox Bright doesn't work anymore. Stamen Terrain works great, and you will see it creates a beautiful terrain map.

**Installing the Library**

If you haven't installed OpenCV yet, please do so by following the instructions below.  If you don't know if you have OpenCV, please open Python and type import cv2. If you don't get an error, it means OpenCV is installed.

**To install:**

1. Open the command line and type:

pip install opencv-python

2. Then open a Python session and try:

import cv2

3. If you get no errors, that means you installed OpenCV successfully. If you get an error, please see the FAQs below:

**FAQs**

**1. My OpenCV installation didn't go well on Windows**

Solution:

1. Uninstall OpenCV with:

pip uninstall opencv-python

2. Download a wheel (.whl) file from [this link](http://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv) and install it with pip. Make sure you download the correct file for your Windows version and your Python version. For example, for Python 3.6 on Windows 64-bit, you would do this:

pip install opencv\_python‑3.2.0‑cp36‑cp36m‑win\_amd64.whl

3. Then try to import cv2 in Python again. If there's still an error, then please type the following again in the command line:

pip install opencv-python   
4. Now, you should successfully import cv2 in Python.

**2. My OpenCV installation didn't go well on Mac**

Solution:

If pip install OpenCV-python didn't go well, please install OpenCV for Python 2 and use Python 2 to run the programs that contain cv2 code. Its' worth mentioning that Python 2 is installed by default on Mac, so no need to install Python 2. Here are the steps to correctly install OpenCV:

1. Install brew:

Open your terminal and execute the following:

/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

2. OpenCV depends on GTK+, so please install that dependency first with brew (always from the terminal):

brew install gtk+

3. Install OpenCV with brew:

brew install opencv

4. Open Python 2 by typing:

python

5. Import cv2 in Python:

import cv2

If you get no errors, that means you installed OpenCV successfully.

**3. My OpenCV installation didn't go well on Linux**

1. Please open your terminal and execute the following commands one by one:

1. sudo apt-get install libqt4-dev
2. cmake -D WITH\_QT=ON ..
3. make
4. sudo make install

2. If that doesn't work, please execute this:

1. sudo apt-get install libopencv-\*

3. Then install OpenCV with pip:

pip install opencv-python

4. Import cv2 in Python. If you get no errors, that means you installed OpenCV successfully.

Resources for this lecture

**Solution: Batch Image Resizing**

import cv2  
import glob

images=glob.glob("\*.jpg")

for image in images:  
    img=cv2.imread(image,0)  
    re=cv2.resize(img,(100,100))  
    cv2.imshow("Hey",re)  
    cv2.waitKey(500)  
    cv2.destroyAllWindows()  
    cv2.imwrite("resized\_"+image,re)

I first created a list containing the image file paths and then iterated through the aforementioned list.

The loop: reads each image, resizes, displays the image, waits for the user input key, closes the window once the key is pressed, and writes the resized image. The name of the resized image will be "resized" plus the existing file name of the original image.