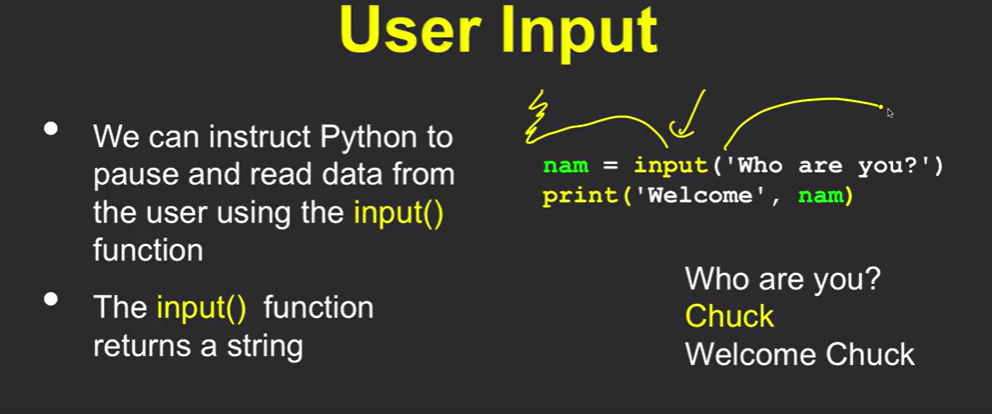
## Getting started with python:

Print function is used to print any outputs

Input function is used to read input from user



[**https://eng.libretexts.org/Bookshelves/Computer\_Science/Programming\_Languages/Python\_for\_Everybody\_(Severance)**](https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_Languages/Python_for_Everybody_(Severance))

to run the python on cmd > use command py or pyhton <file.py>.

Reserve words

variables

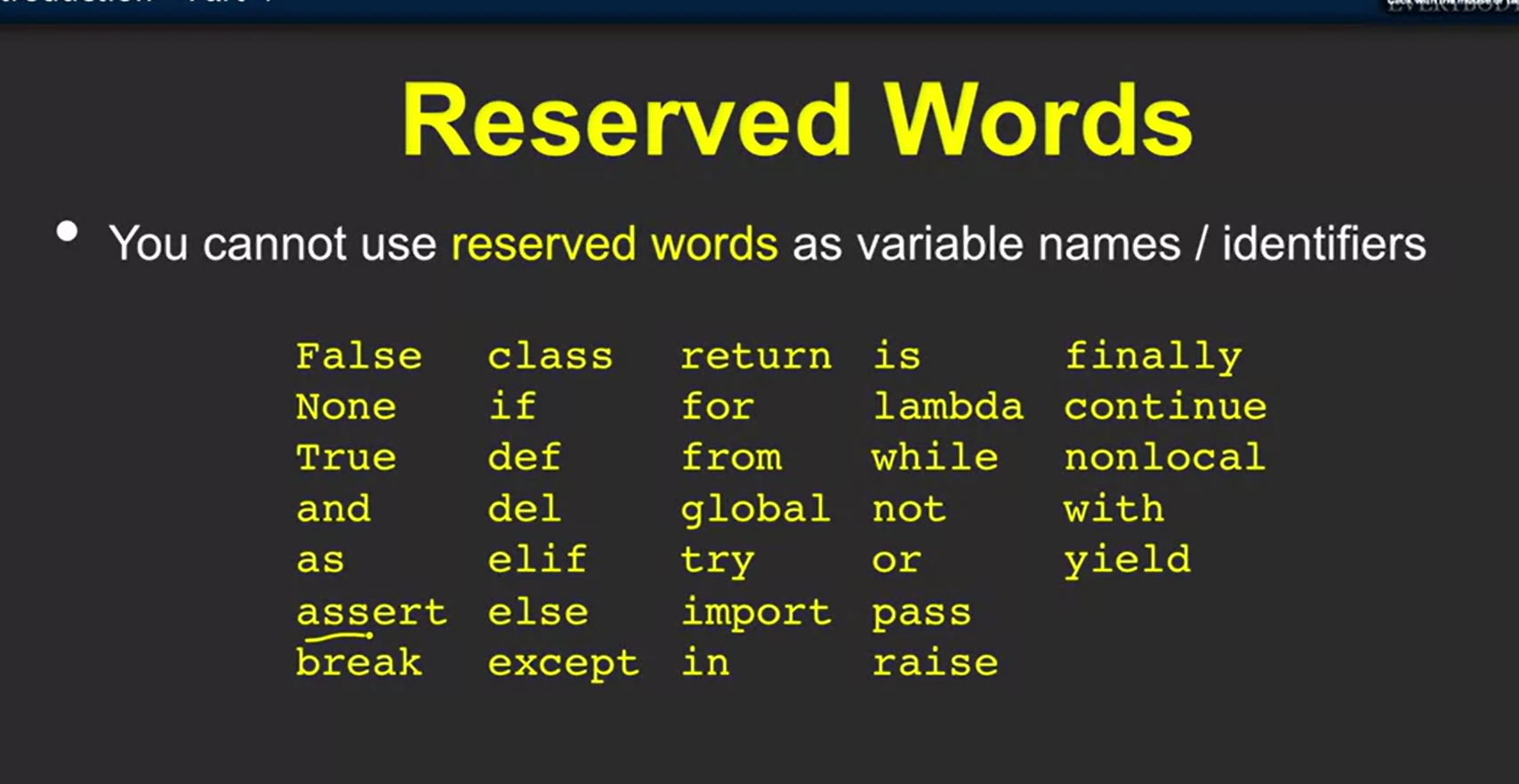
String and integer conversion

Conditional statements

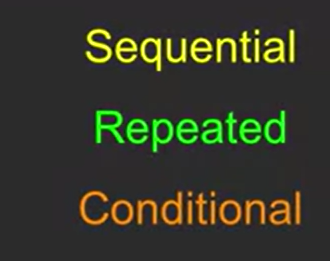
w

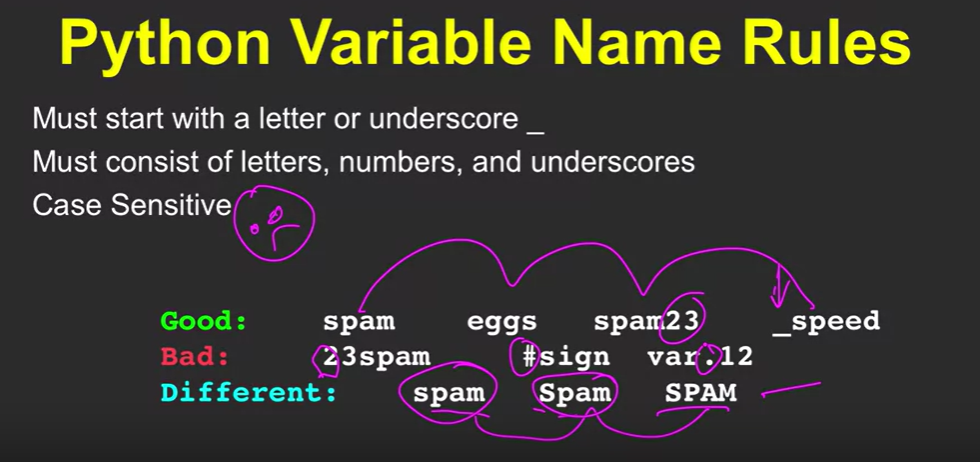
Control flow , Algorithms, and Data structures

### Reserve words in python

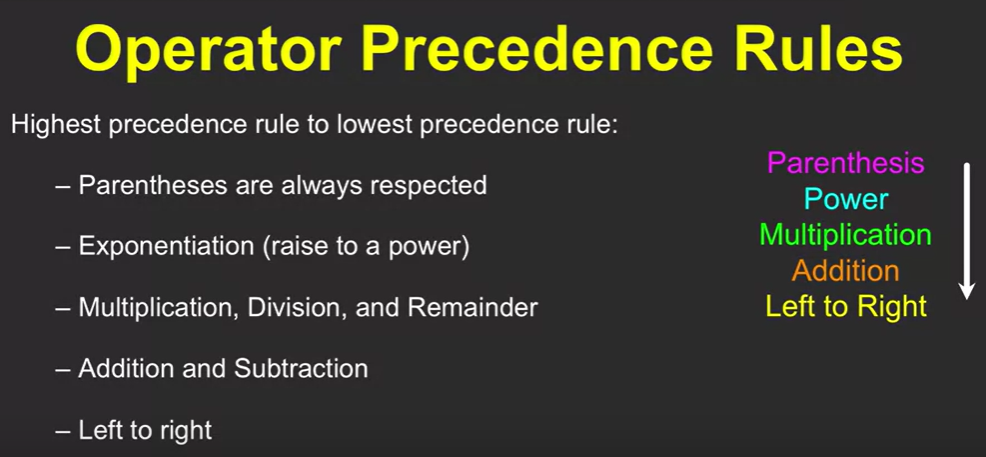


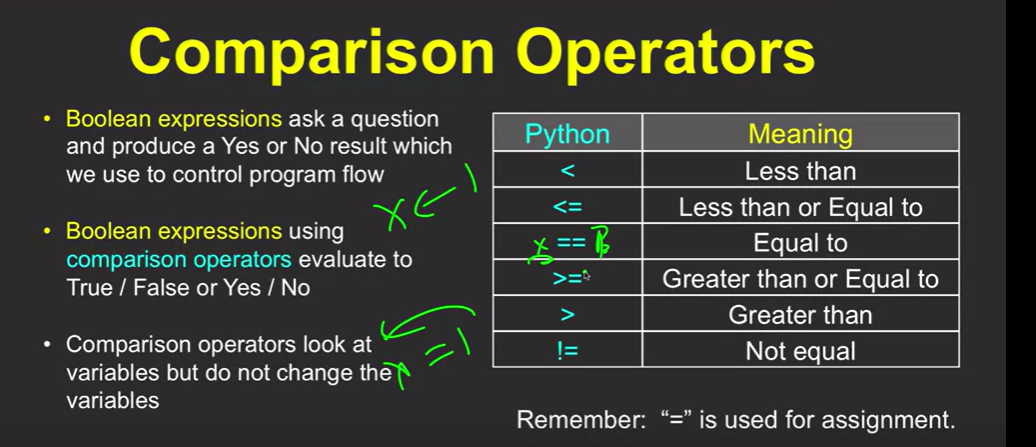
Types of syntax in any programming



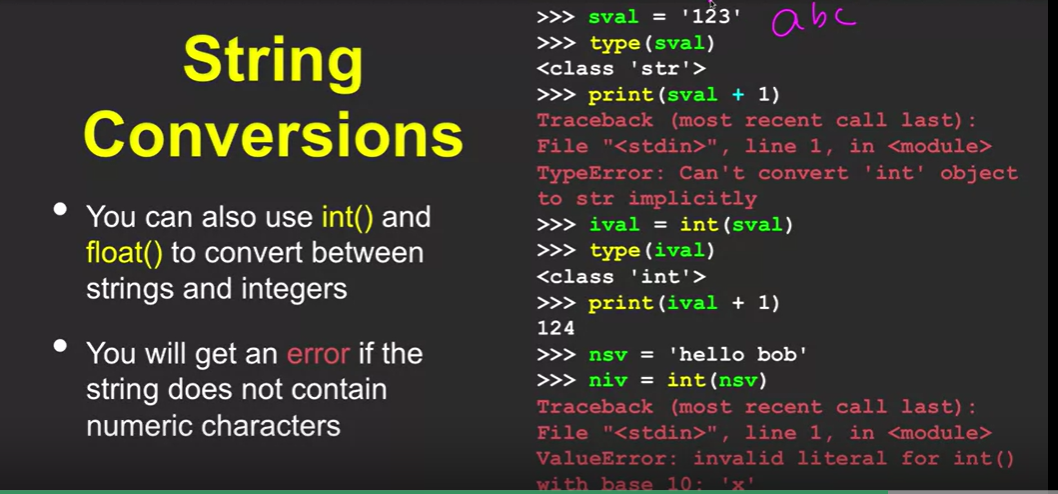
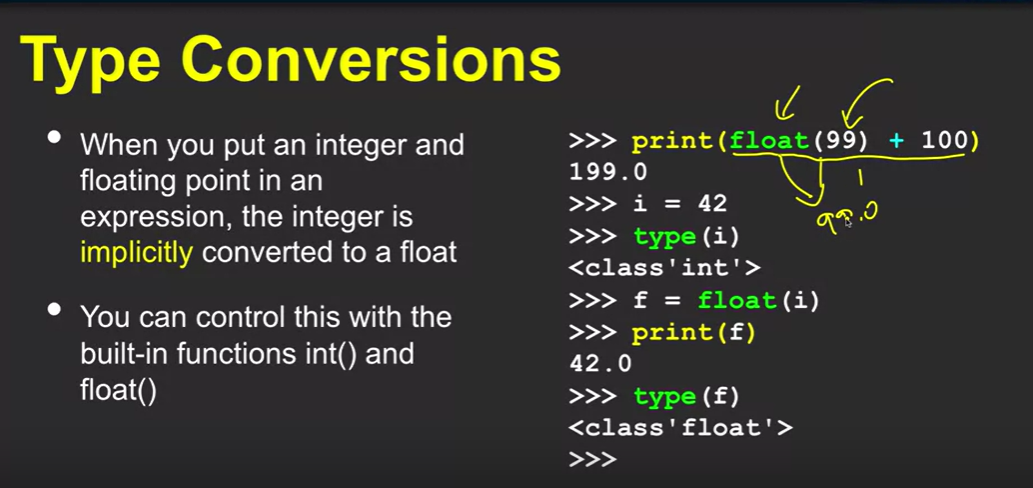


Mnemonic is technique to name the variable





Indentation matters in loops at python



Error Handling in Python with

try: and except:

when error occurs in try block the except will get execute. Until unless error in try block then only except block runs

### Functions:

Store and reuse

def reserve word is used to define the function and return keyword is used to return the output.

Must match the type of parameter and number of inputs while passing an argument to the function

If the invoking/calling a function done without parentheses will return the function definition, while invoking a function with parentheses will compute the function.

**With out parentheses:**

pf = computepay

p =pf(hrs,rate)

**With parentheses:**

p = computepay(hrs, rate)

print("Pay", p)

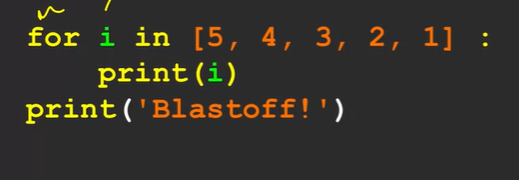
### Condition, Loops, and Iteration:

Condition is piece of comparison operator expression, works based on the true/false

Where humans are poor at repeating computation, can achieve this using programming loop technique.

break and continue

Definite loops



**for** i **in** range(0, n):

    print(i)

Can iterate through the lists[...], tuples(...), strings, dictionaries are like maps, and sets{...}

### **Python packages**

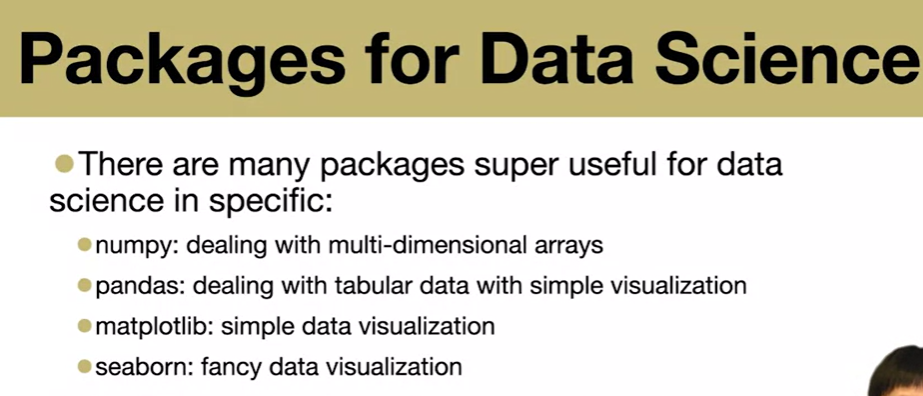
* import random – provides randint(0,n) e.t.c functions – random module

randint(0,10)- to generate random integers from 1 to 10

random() – to generate a random float

uniform(0,10) -to generate a random flaot between 0 to 10

* import math – math module



#### **Python Random Module:** This code uses the random module to select a random element from the list list1 using the **random.choice()**function. It prints a random element from the list, demonstrating how to pick a random item from a sequence in Python.

**import** random

list1 **=** [1, 2, 3, 4, 5, 6]

print(random.choice(list1))

The seed() seed makes these numbers the same every time you run the code with a seed of 5, providing consistency in the generated random values.

**import** random

random.seed(5)

print(random.random())#gerates same number for every time

print(random.random())#gerates same number for every time

[random.randint()](https://www.geeksforgeeks.org/python-randint-function/) method is used to generate random integers between the given range.

A [random.random()](https://www.geeksforgeeks.org/random-random-function-in-python/) method is used to generate random floats between 0.0 to 1.

A [random.shuffle()](https://www.geeksforgeeks.org/random-shuffle-function-in-python/) method is used to shuffle a sequence (list). Shuffling means changing the position of the elements of the sequence. Here, the shuffling operation is in place.

**Syntax:** random.shuffle(sequence, function)

A random.uniform(a,b) menthod Returns the generated floating point random number between lower limit and upper limit

sample() - random.sample(iterable, k: int)

**sample** returns a list with a random selection from an iterable. The number of elements returned is equal to the **k** parameter:

>>> random.sample([1, 2, 3, 4], 1)

# [3]

>>> random.sample([1, 2, 3, 4], 2)

# [3, 4]

### ***Python Math Module:***

math.e - The **math.e** constant returns the Euler’s number: 2.71828182846.

math.pi - The pi is depicted as either 22/7 or 3.14. **math.pi** provides a more precise value for the pi.

[**Tau**](https://www.geeksforgeeks.org/tau-mathematical-constant/) is defined as the ratio of the circumference to the radius of a circle. The **math.tau** constant returns the value tau: 6.283185307179586.

The Python **math.inf** constant returns of positive infinity. For negative infinity, use **-math.inf**.

The Python **math.nan** constant returns a floating-point nan (Not a Number) value. This value is not a legal number. The nan constant is equivalent to float(“nan”).

#### Numeric Functions

**import** math

Ceil value means the smallest integral value greater than the number and the floor value means the greatest integral value smaller than the number. This can be easily calculated using the [**ceil()**](https://www.geeksforgeeks.org/python-math-ceil-function/) and [**floor()**](https://www.geeksforgeeks.org/python-math-floor-function/) method respectively.

Using the [**factorial()**](https://www.geeksforgeeks.org/python-math-factorial-function/)function we can find the factorial of a number in a single line of the code. An error message is displayed if number is not integral.

[**gcd()**](https://www.geeksforgeeks.org/python-math-gcd-function/)function is used to find the greatest common divisor of two numbers passed as the arguments. print (math.gcd(b, a))

[**fabs()**](https://www.geeksforgeeks.org/python-math-fabs-function/) function returns the absolute value of the number.

#### **Logarithmic and Power Functions**

[exp()](https://www.geeksforgeeks.org/python-math-library-exp-method/) method is used to calculate the power of eor we can say exponential of y.

[**pow()**](https://www.geeksforgeeks.org/pow-in-python/) function computes x\*\*y. This function first converts its arguments into float and then computes the power.

[**sqrt()**](https://www.geeksforgeeks.org/python-math-function-sqrt/) function returns the square root of the number.

Finding the Logarithm

* **log()** function returns the logarithmic value of a with base b. If the base is not mentioned, the computed value is of the natural log.
* **log2(a)**function computes value of log a with base 2. This value is more accurate than the value of the function discussed above.
* **log10(a)** function computes value of log a with base 10. This value is more accurate than the value of the function discussed above.

**sin(), cos(), and tan()** functions returns the sine, cosine, and tangent of value passed as the argument. The value passed in this function should be in **radians**.

Converting values from degrees to radians and vice versa

* **degrees()** function is used to convert argument value from radians to degrees.
* **radians()** function is used to convert argument value from degrees to radians.

**math.isinf()** function is used to check whether the value is infinity or not.

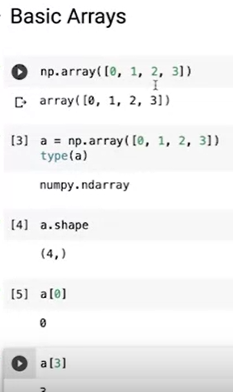
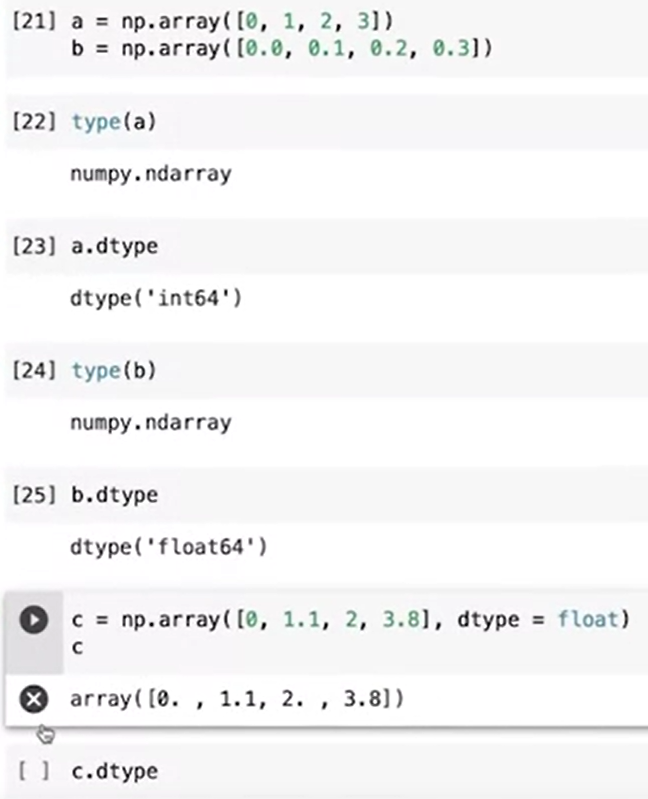
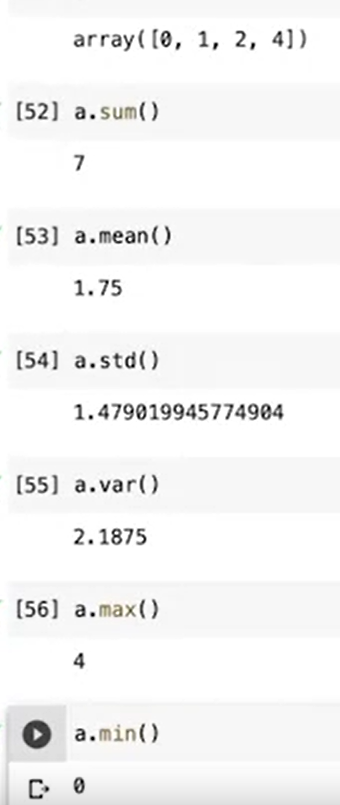
### **Python Numpy – Data manipulation**

Numerical python

Uses array data structure

To convert list to NumPy array, Numpy is pyhton module to compute the array data structure

**Numpy** is a package to deal with multi-dimensional data

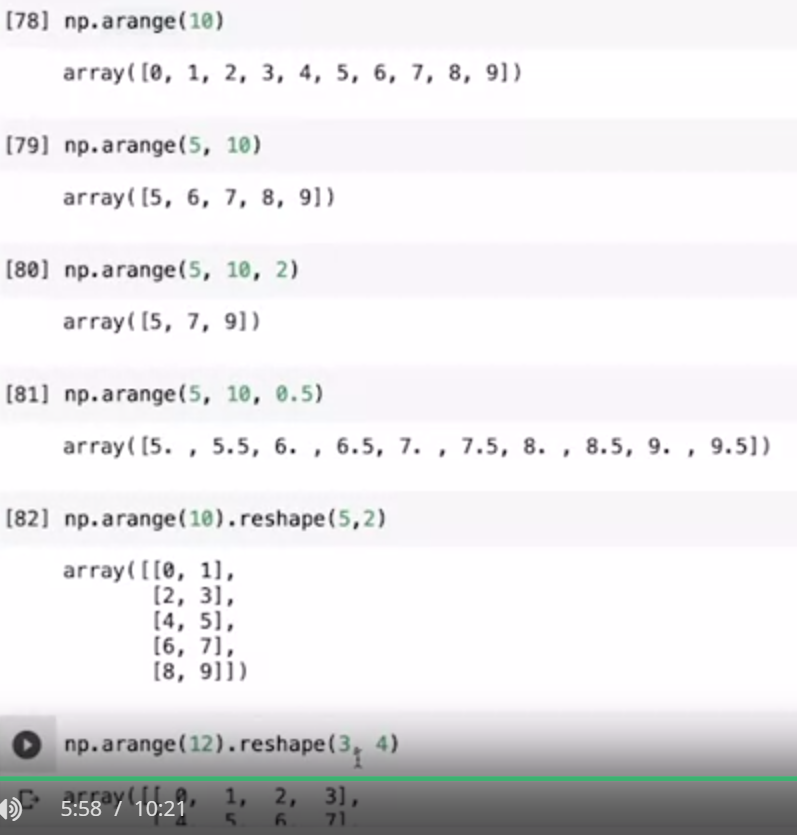
Slicing in array

array[0:] (forward slicing)– the zero is index where it starts the slicing

array[:1] (backward slicing)– the one is index(exclusive) where it starts to backword the slicing

**Linear algebra:**

Addition, subtraction, and multiplication of arrays



### **Python Pandas Module:**

Import pandas as pd – A DataFrame, uses the DataFrames and Series data structure

**Pandas** is a package to deal with tabular data in particular, the DataFrame.

Creating data frame with an object properties can form a table of data as output – df.info, df.index, df.size, df.shape, df.dtypes

Can access the specific column by df[‘{columnname}’]



### Python Matplotlib Module:

Data visualization - import matplotlib as plt

Plot function or scatter function - to plot graph between x and y

colorbar() function

plt.plot(x, y)t

plt.plot(x, y, 'o') – the ‘o’ represents the marker type

markers = ['o', '\*', '.', ',', 'x', 'X', '+', 'P', 's', 'D', 'd', 'p', 'H', 'h', 'v', '^', '<', '>', '1', '2', '3', '4', '|', '\_']

plt.plot(y, '1')

line = ['-',':','--','-.']

plt.plot(x\*2,y,'--')

color = ['r','g','b','c','m','y','k','w']

c = 'o:' + c

plt.plot(y, c)

plt.plot(z, c, label = 'snb')

plt.grid()

plt.legend(['snb'])

Scatter, line, bar, pie, histogram, box,

plt.scatter(x, y, c = colors, cmap = 'hot', alpha = 0.5, s = sizes)- marks only points

plt.colorbar()

plt.bar(x, y)

plt.hist(x)

plt.pie(x, labels = ['2/3', '1/5', 'others'])

### Python Seaborn Module:

Import seaborn as sns

From data load to data visualization.

Pandas+matplotlib

Relation plots:

sns.relplot(data = tips, x = 'total\_bill', y = 'tip')

sns.relplot(data = tips, x = 'total\_bill', y = 'tip', hue = 'sex')

sns.relplot(data = tips, x = 'total\_bill', y = 'tip', hue = 'day')

sns.relplot(data = tips, x = 'total\_bill', y = 'tip', hue = 'time')

sns.relplot(data = tips, x = 'total\_bill', y = 'tip', hue = 'smoker', col = 'time')

sns.relplot(data = tips, x = 'total\_bill', y = 'tip', hue = 'size')

sns.relplot(data = tips, x = 'total\_bill', y = 'tip', size = 'size', hue = 'size')

sns.relplot(data = tips, x = 'total\_bill', y = 'tip', size = 'size', sizes = (15, 200), hue = 'size')

sns.relplot(data = tips, x = 'total\_bill', y = 'tip', size = 'size', sizes = (15, 200), alpha= 0.5, hue = 'size')

Distributional plots:

sns.displot(data = tips, x = 'total\_bill')

sns.displot(data = tips, x = 'total\_bill', col = 'sex')

sns.displot(data = tips, x = 'total\_bill', hue = 'sex')

sns.displot(data = tips, x = 'total\_bill', hue = 'sex', col = 'day')

sns.displot(data = tips, x = 'total\_bill', hue = 'sex', kind = 'kde')

sns.displot(data = tips, x = 'total\_bill', hue = 'sex', kind = 'kde', multiple = 'stack')

sns.displot(data = tips, x = 'total\_bill', hue = 'sex', kind = 'kde', col = 'day', multiple = 'stack')

Categorical Plots:

sns.catplot(data = tips, x = 'day', y = 'total\_bill')

sns.catplot(data = tips, x = 'day', y = 'total\_bill', kind = 'box')

sns.catplot(data = tips, x = 'day', y = 'total\_bill', kind = 'box')

sns.catplot(data = tips, x = 'day', y = 'total\_bill', kind = 'box', hue = 'sex')

sns.catplot(data = tips, x = 'day', y = 'total\_bill', kind = 'violin')

sns.catplot(data = tips, x = 'day', y = 'total\_bill', kind = 'violin', hue = 'sex')

sns.catplot(data = tips, x = 'day', y = 'total\_bill', kind = 'violin', hue = 'sex', split = True)

sns.violinplot(data = tips, x = 'day', y = 'total\_bill', hue = 'sex', split = True, inner = 'quartile')

sns.catplot(data = tips, x = 'day', y= 'total\_bill', kind = 'bar')

Joints Plots:

sns.jointplot(data = tips, x = 'total\_bill', y = 'tip', hue = 'sex', kind = 'kde')

sns.jointplot(data = tips, x = 'total\_bill', y = 'tip', hue = 'sex', kind = 'scatter')

kind = ['scatter', 'hist', 'hex', 'kde', 'reg', 'resid']

for k in kind:

  sns.jointplot(data = tips, x = 'total\_bill', y = 'tip', kind = k)

Pair Plots:

sns.pairplot(data = tips)

sns.pairplot(data = tips, hue = 'sex')

**Here are several effective methods to run script files when** main.py **is executed:**

**1. Direct Execution within** main.py**:**

* Use the import statement to bring the script's functions and classes into main.py.
* Call those functions or use those classes directly within main.py.

**Example:**

Python

# main.py

import my\_script # Assuming script is in a file named "my\_script.py"

my\_script.my\_function()

result = my\_script.my\_class()

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

**2. Function Calls within** main.py**:**

* Define functions in your script file that perform the desired actions.
* Import those functions into main.py and call them as needed.

**Example:**

Python

# my\_script.py

def perform\_task():

# Task code here

# main.py

import my\_script

my\_script.perform\_task()

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

**3. Subprocess Execution:**

* Use the subprocess module to run the script as a separate process.
* Pass arguments and capture output as needed.

**Example:**

Python

# main.py

import subprocess

subprocess.run(["python", "my\_script.py", "arg1", "arg2"]) # Pass arguments if required

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

**4. Module Initialization:**

* Add code to your script file that executes upon import.
* Import the script within main.py to trigger its execution.

**Example:**

Python

# my\_script.py

def init():

# Code to execute when imported

init() # Execute init() upon import

# main.py

import my\_script # Triggers init() execution

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

**Choosing the Right Method:**

* **Direct execution** and **function calls** are best for tight integration and shared variables.
* **Subprocess execution** is useful for independent execution or resource isolation.
* **Module initialization** is convenient for automatic setup or configuration tasks.

**Additional Considerations:**

* **Script Location:** Ensure scripts are in a directory accessible to Python's import path.
* **Argument Passing:** Use sys.argv to access command-line arguments passed to main.py.
* **Error Handling:** Implement proper error handling within scripts and in main.py.
* **Modularity:** For complex projects, create separate modules or packages for better organization.