**Data Structures**

int: **1, 100, -45**

float: Floating-point type, for representing numbers with a decimal point (e.g., **1.0, 3.1415**)

bool: **True or False**

str: **"Hello", "123"**

None: Special type representing the absence of a value or a **null value** in Python

**FUNCTIONS**

* [::2] means take every 2nd letter abcdefg would be aceg slice is start:stop:step
* Ditc has keys a and b. u go print(“z” in data) the result is False
* Append([4,5]) will literally append the list to a list ok, to add elements individually use extend()
* Enumerate(a list) – it takes the index and the value so u can go for index, value in enumerate(a list)
* Zip(list1,list2) – first u need to do list(zip(..)) and it pairs the elements one by one so if list1 = ‘x’,’y’ and list2=’e’,’r’,’t’ then u’ll get [(‘x’,’e’), (‘y’,’r’)] , course u can have more than 2 lists
* // = floor division (ex. 7 // 2 = 3 so 3.5 goes down to 3)
* % = reminder (ex. 7 % 2 = 1
* float(), int(), str(), list(), type()
* % -> rest la impartire 22%3 = 1
* round(int, [nr. of decimals]) round(3.5555,2)
* range(start, stop, step) range(3) = [0,1,2]
* print(f"Hello, {name}") print(hello, name) –> “hello name”
* If, elif, else
* For, break, while
* Or, and, ==, !=
* Lambda x: x\*\*2
* Def, return
* list(zip(lista1,lista2)) – zipu iti face un fel de dict da iti tre list sa dai print()
* varible = input(“Gimme smth”)
* match name (a variable): case “smth” (obv multiple case): print(“idk”)
* while True:

try: x = int(input("Enter a number: "))

except ValueError(an error):

print(“the reason for the error”) OR pass (it goes straight to relooping)

* random.choice([“heads”, “tails”])
* calss Students:
  + def \_\_init\_\_(self, name, house):
    - self.name = name
    - self.house = house
* def\_\_str\_\_(self):
  + return f”{self.name} from {self.house}”
* if house not in [“house1”, “house2”]:
  + raise ValueError(“Invalid house”)

**Common syntax**

This goes at the end of the code (why? To prevent form running main if it was imported as a module)

if\_\_name\_\_ == “\_\_main\_\_”:

main()

def(x:int) -> str or None(if the functions prints like doesn’t return a value it does a command and done)

**LIST**

**ORDERED + MUTABLE + DUPLICATES OK** (e.g., [1, 2, 3] or ['a', 'b', 'c']).

List = [1, 2, 3, 4, 5, 6, 7, 8, 9]

* list("abc") → ['a', 'b', 'c']
* List[0] = 1, List[-1] = 9
* List[0:4] = (0,1,2,3) List[-3:0] = (7,8,9)
* List.insert(index, obj)
* List.index(obj) – returns the first occurrence index
* List.append(obj) – goes to the end of the list
* List.remove(obj) – removes the first occurrence
* List.pop(index) – last elem if no index
* List.count(smth) – how many times smth is in the list
* List.sort() – sorts in ascending order
* Variable = List.copy() – self explanatory
* List.reverse()
* list(map(lambda x: x \* 2, List)) (list face lista, map aplica lambda la tate nr din List)
* list(filter(lambda x: x % 2 == 0, List)) (list face lista, filter aplica lambda la tate nr din List si selecteaza)
* len(List) = nr of obj in List
* List.clear()
* a.extend(b) unde a = [1, 2], b = [3, 4], a = [1, 2, 3, 4]

**TUPLES**

**ORDERED + IMMUTABLE + U CAN DO THIS a,b,c = (1,2,3)** (e.g., (1, 2, 3)).

* Tuple = (1, 2, 3, 4, 5, 6)
* Tuple.count(obj)
* Tuple.index(obj)
* (1, 2) + (3,) → (1, 2, 3)

**SETS**

**UNORDERED + NO DUPLICATES + MUTABLE**  (e.g., {1, 2, 3}).

Set = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, Set2 = {0, 1, 2, 3, 11, 12, 13, 14, 15}

* Set.add(100)
* Set.update(Set2)
* Set.union(Set2)
* Set.intersection(Set2)
* Set.difference(Set2) – {4,5,6,7,8,9}
* Set.symmetric\_difference(Set2) – all elem that u can’t get in both sets (inversu intersectiei)
* Set.issubset(Set2) – True/False
* Set.issuperset(Set2) – True/False
* Set.isdisjoint(Set2) – True/False (do they have some element in common? If NO – > True else False)
* Set.remove(obj), Set.discard(obj) – a 2 a nu raise any error if obj doesn’t exist

**STRINGS**

Text = “This, is a string ” ; text2 = '...+...lemons and limes...-...' ; str = “Yellow” ; words = [‘aur’, ‘ar’]

* " ".join(["hello", "world"]) -> 'hello world'
* str[0]=”Y”
* Text.split() -> [“This”, “is”, “a” , “string”]
* Text.strip() -> “This is a string”
* Text.split(“delimiter”) -> ex. Text.split(“,”) = [“This”, “is a string”]
* Text.strip() -> eliminate space left and right so Text = “This, is a string”
* text2.strip('.+-') => 'lemons and limes'
* Text.title() – “ This, Is A String”
* Text.lower() – “ this, is a string”
* Text.upper() – “ THIS, IS A STRING”
* Text2.capitalize() – '...+...Lemons and limes...-...'
* Text.find(“\_”) – nr of times u get the \_
* variable = “, ”.join(words) -> “aur, ar”
* Text.find(“is”) -> 6 (the index the word starts with i
* Text.count(“is”) -> 1
* Text.isalpha() – Checks if all characters are letters (TRUE/FALSE)
* Text.isdigit() – Checks if all characters are digits (TRUE/FALSE)
* Text.isalnum() – Checks if all characters are letters or digits (TRUE/FALSE)
* Text.replace(smth there is, with smth u want, first/2nd/.. occurance of that which it is)

**DICTIONARIES**

**UNORDERED + MUTABLE + UNIQUE KEY** (e.g., {'key': 'value', 'id': 1}).

Students = {

“Hermione”:”Gryffindor”

“Hermione1”:”Gryffindor1”

}

Dict = {key: value, key2: value2}, Dict2 = {altekeie: “ceva”, keiapaci: (“hat”, 34)}

* Dict[“key”] = value
* Dict.get(key, “N-are valoare”) -> valoarea cheii ori al 2lea argument
* Dict.pop(key) -> ramai cu Dict = {key2: value2} si poti salva gen fa variable = Dict.pop(key)
* Dict.popitem() – la fel ca pop numa sti ca ii ultima pereche the keie,valoare introdusa (introdusa, da)
* Dict.keys() – access just keys (returns a dict obj, behaves like a list but is not a list)
* Dict.values() – access just values (returns a dict obj, behaves like a list but is not a list)
* Dict.items() – iti da (keye, valoare) ca un obiect
* Dict.setdefault(key, default) -> Returns the value if key exists, else sets it.
* Dict.update(Dict2)
* Dict.clear()
* {x:x\*\*2 for x in range(5)} – smecher
* Dict.copy() – mai mult pt Dict.update(Dict2) ca iti schimba Dict
* d1 | d2 is the dict merge operator

**ERRORS**

**SyntaxError**: Ex. print("Hello" – missin the ) at the end

**NameError**: Ex. print(x) but x not defined earlier

**TypeError**: Ex. print("Age: " + 25) – can’t use + like str + int ….

**IndexError**: Ex. You’re trying to access an item that doesn’t exist in a list.

**KeyError**: Ex. You’re trying to access a key in a dictionary that doesn’t exist.

**ValueError**: Ex. int("hello") – Ensure the value is a number before converting

**AttributeError**: Ex.

num = 10

num.append(5)

'int' object has no attribute 'append'

**PANDAS**

**Creating DataFrames**

df = pd.DataFrame({

'A': [1, 2, 3],

'B': ['x', 'y', 'z'],

'C': [4.5, 5.1, 6.3]

}) ->

Index A B C  
0 1 x 4.5

1 2 y 5.1

2 3 z 6.3

df = pd.read\_csv('file.csv') # Read CSV

df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]}) # From Dict

**Viewing Data**

df.head() # First 5 rows

df.tail(3) # Last 3 rows

df.describe() # Summary statistics

df.shape # (rows, columns)

df.columns # Column names

df.dtypes # Data types

df.isnull().sum() # Missing values count

df.duplicated().sum() # Duplicate rows count

**Selecting & Filtering Data**

df['A'] # Select single column

df[['A', 'B']] # Select multiple columns

df.iloc[0] # First row

df.iloc[:, 1] # First column

df.iloc[1:3, 0:2] # Slicing rows & columns

df.loc[df['A'] > 2] # Filter rows

df[(df.month == "March") | (df.month == "April")]

df.query("A > 2 and B == 'z'") # Query method

**Modifying Data**

df['D'] = df['A'] \* 2 # Add new column

df.drop('B', axis=1) # Drop column

df.drop(2, axis=0) # Drop row; df.drop(1) - remove row with index 1

df.rename(columns={'A': 'Alpha'}) # Rename column

df['C'] = df['C'].fillna(df['C'].mean()) # Fill missing values

df.replace({'x': 'apple', 'y': 'banana'}, inplace=True) # Replace values

df['x'].replace(2, 99) - replace all 2s with 99

df['A'] = df['A'].astype(float) # Change data type

df.at[1, "num"] = 99 - this row this col I want this val

**Sorting & Grouping**

df.sort\_values(by='A', ascending=False) # Sort by column

df.sort\_index() # Sort by index

df.groupby('B').sum() # Group by column

df.groupby(['A', 'B']).agg({'C': 'mean'}) # Multiple groupby with aggregation

**Aggregation & Statistics**

df.mean() # Mean of each column

df.median() # Median

df.std() # Standard deviation

df['A'].value\_counts() # Count unique values

df.corr() # Correlation matrix

**Merging & Joining**

df1.merge(df2, on='key') # Inner join

df1.merge(df2, on='key', how='left') # Left join

df1.merge(df2, on='key', how='right') # Right join

df1.append(df2, ignore\_index=True) # Append rows

df1.concat([df1, df2], axis=1) # Concatenate columns

**Working with Dates**

df['date'] = pd.to\_datetime(df['date']) # Convert to datetime

df['year'] = df['date'].dt.year # Extract year

df['month'] = df['date'].dt.month # Extract month

df['day'] = df['date'].dt.day # Extract day

df.set\_index('date', inplace=True) # Set date column as index

**Exporting Data**

df.to\_csv('output.csv', index=False) # Save as CSV

df.to\_excel('output.xlsx', index=False) # Save as Excel

**NUMPY**

arr[1] # Get element at index 1

arr[:3] # Get first 3 elements

arr2D[1, 2] # Get element at row 1, col 2

arr[:, 1] # Get all rows, column 1

arr = np.array([1, 2, 3]) # 1D array

arr2D = np.array([[1, 2, 3], [4, 5, 6]]) # 2D array

arr + 2 # Add 2 to each element

arr \* 3 # Multiply each element by 3

arr1 + arr2 # Element-wise addition

np.dot(arr1, arr2) # Matrix multiplication

np.linspace(0, 10, 5) # 5 numbers from 0 to 10

plt.plot(x, y, label="sin(x)", color='r', linestyle="--")