PYTHON CHEAT SHEET

FUNCTIONS

Functions help in encapsulating logic and prevent us from using the same piece of code again and again

Description	Syntax	Example

Defining a function in python	def name_of_function():
	```function body```
	
	# calling the function
	name_of_function()
		def make_tea():
		print("i am making tea")
		
		make_tea()
		OUTPUT: i am making tea

Functions can take in parameters that can be used within a string body.	def name_of_function(param): ```function body```	def print_name(name): print("my name is", name print_name("rob") OUTPUT: my name is rob
The "return" keyword from within the function can be used to return a value after a function call.	def name_of_function(x, y): # do something return z	def return_sum(x, y): z = x + y return z my_sum = return_sum(1, 2) print(my_sum) OUTPUT: 3

TYPE OF ARGUMENTS IN FUNCTIONS

Positional Arguments	def random(a, b, c, d): print(b, c, a, d) random(4, 5, 6, 7) OUTPUT: 5 6 4 7
Keyworded Arguments	def random(a, b, c, d): print(b, c, a, d) random(a = 4, b = 5, c = 6, d = 7) OUTPUT: 4 5 6 7
All Keyword Arguments Must Follow All Positional Arguments	Wrong Syntax random(B = 4, 5, 6, 7) > Error Correct Syntax random(4, 5, C = 6, D = 7) OUTPUT: 4 5 6 7

Scope Of Variables	local variable inside the function is present
	a = 10 # global variable
	def random():
	a = 50 # local variable
	print(a)
	
	random()
	OUTPUT: 50
	
	local variable inside the function is not present
	a = 10 # global variable
	def random():
	print(a)
	
	random()
	OUTPUT: 10

"Global" keyword in front of a variable name is used to tell python to use the global value of that variable.	a = 10 def random(): global a a = 20 print(a) random() print(a) OUTPUT: 20 20

LISTS

Creating a list	list_name = [element_1, element_2, …] Example numbers = [1, 2, 3, 4, 5] print(numbers) OUTPUT: [1, 2, 3, 4, 5]

ACCESSING ELEMENTS FROM A LIST

Description	Syntax	Example
Indexing a list to access a value. the index number ranges between 0 and the length of the list.	list_name[index_number]	nums = [1, 2, 3] print(nums[0]) OUTPUT: 1 print(nums[2]) OUTPUT: 3 print(nums[3]) OUTPUT: INDEXERROR
Negative indexing to access elements from the end.	list_name[index_number]	nums = [1, 2, 3] print(nums[-1]) OUTPUT: 3

METHODS TO ADD VALUES TO A LIST

Description	Syntax	Example
List.append(): adds an element at the end of a list	list_name.append(element)	nums = [1, 2, 3] nums.append(4) print(nums) OUTPUT: [1, 2, 3, 4]
List.insert(): add an element at a specific index in the list.	list_name.insert(index_number, element_to_insert)	nums = [1, 3, 4] nums.insert(1, 2) print(nums) OUTPUT: [1, 2, 3, 4]
List.extend(): add multiple values at the end of a list.	list_name.extend(list_of_values)	nums = [1] nums.extend([2, 3, 4]) print(nums) OUTPUT: [1, 2, 3, 4]

LIST SLICING

Used for accessing a range of elements from a list.

Description	Syntax	Example
Accessing a range of **elements**	list_name[start_idx : end_idx]	nums = [1, 2, 3, 4, 5] slice = nums[1:3] print(slice) OUTPUT: [2, 3]
Accessing values at **odd indexes.**	list_name[start:end:step_size]	nums = [1, 2, 3, 4, 5] slice = nums[0:len(nums):2] print(slice) OUTPUT: [1, 3, 5]
Accessing all **elements** from a **particular index till the end.**	list_name[start:]	nums = [1, 2, 3, 4, 5] slice = nums[2:] print(slice) OUTPUT: [3, 4, 5]
Accessing all the **elements** from the **start till** the **specified index**	list_name[:end]	nums = [1, 2, 3, 4, 5] slice = nums[:3] print(slice) OUTPUT: [1, 2, 3]
A step size of -1 can be used to traverse a list from the back. (reversed list)	list_name[::-1]	nums = [1, 2, 3, 4, 5] slice = nums[::-1] print(slice) OUTPUT: [5, 4, 3, 2, 1]

Iterating over a list

Description	Syntax	Example

Iterating using a for loop with the **range function.**	**for** i **in** range(**len**(list_name)):
	  element = list_name[i]
	  ```do something```
		nums = [1, 2, 3, 4, 5]
		for i in range(**len**(nums)):
		  print(nums[i], end = " ")
		OUTPUT: 1 2 3 4 5

Iterating using a for loop **without range.**	**for** x **in** list_name:
	  ```do something```
		nums = [1, 2, 3, 4, 5]
		for x in nums:
		  print(x, end = " ")
		OUTPUT: 1 2 3 4 5

Few more list methods

Description	Syntax	Example
List.pop() removes an element at a particular index in a list. If value no index passed → last value is popped.	list_name.pop(index_to_pop)	nums = [1, 2, 3] nums.pop() print(nums) OUTPUT: [1, 2] nums.pop(0) print(nums) OUTPUT: [1, 3]
List.remove() removes a particular value from the list. Multiple occurrences → removes the first occurrence If value not present → error.	list_name.remove(value_to_remove)	nums = [1, 2, 3] nums.remove(3) print(nums) OUTPUT: [1, 3]
List.index() returns the first index of a value in the list that is passed in as a parameter to the method. Multiple occurrences → removes the first occurrence If the value is not present → error.	list_name.index(value)	nums = [1, 2, 3] idx =nums.index(2) print(idx) OUTPUT: 1
List.count() returns the number of times a value passed to the method occurs in the list. If value not present → returns 0	list_name.count(value)	nums = [1, 2, 2, 3] c = nums.count(2) print(c) OUTPUT: 2

2D Lists

Description	Syntax	Example
Initializing a 2d list	list_name = [sub_lists1, sublist2, …]	mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]
Indexing a 2d list. we need two index values for indexing, one for the outer list and one for the inner list.	list_name[outer_idx][inner_idx]	mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9] print(mat[1][2]) OUTPUT: 6 print(mat[0][0]) OUTPUT: 1

Strings

Type of arguments in Functions

Strings are a sequence of characters

Description	Syntax	Example
Defining a string in python	string_name = "string_content"	my_name = "rob" print(my_name) OUTPUT: rob
Concatenating two strings → the "+" operator can be used to concatenate(join) two strings.	new_str = string_1 + string_2	first_name = "rob" last_name = "meyer" print(first_name + " " + last_name) OUTPUT: rob meyer
Strings are compared using lexicographical order (dictionary order) • Note: "a" < "a"	string_1 > string_2 string_1 < string_2 string_1 == string_2	my_name = "rob" your_name = "corn" print(my_name > your_name) OUTPUT: true print(my_name < your_name) OUTPUT: false print(my_name == your_name) OUTPUT: false
Ord() eturns ascii value of a character.	ord(character)	ascii_val = ord("a") print(ascii_val) OUTPUT: 97
Chr() eturns the character mapped to the ascii passed	chr(ascii_value)	character = chr(23123) print(character) OUTPUT: 彳

String methods

Description	Syntax	Example
Str.split() splits a string based on a certain separator	string_name.split(separator)	ex = "this#is#a#string#" splitted = ex.split("#") print(splitted) OUTPUT: ["THIS", "IS", "A", "STRING"]
Str.join() joins a list of strings based on a separator	separator_string.join(list_of_strings)	print(" ".join(["list", "of", "strings"])) OUTPUT: LIST OF STRINGS
Str.replace() replaces a certain occurrence of the first character with a second character	string_name.replace(char1, char2)	newstr = "this is str".replace(" ", "_") print(newstr) OUTPUT: THIS_IS_STR
Str.find() finds the exact sequence or substring in the original string and returns the starting index of the substring. if sequence not present → returns -1	string_name.find(sequence)	s = "this is str" print(s.find("is")) OUTPUT: 2
Str.count() counts the number of times a character or substring is present within a string.	string_name.count(sequence)	s = "this is str" print(s.find("t")) OUTPUT: 2

Tuples

A tuple is like an immutable list

Description	Syntax	Example
Declaring a tuple	tuple_name = (elem1, elem2,…)	t = (1, 2, 3, 4)

Elements of a tuple are immutable.	---	`t = (1, 2, 3, 4)`
		`t[0] = 2`
		`OUTPUT: ERROR`

Positive and negative indexing including slicing is permitted on tuples.	`tuple_name[start:end]`	`t = (1, 2, 3, 4)`
		`print(t[2:8])`
		`OUTPUT: (3 ,4)`
		`print(t[-2:-8])`
		`OUTPUT: ()`

Declaring a tuple with a single element.	`tuple_name = (elem1,)`	`t = (1)`
		`print(t)`
		`OUTPUT: 1`
		`t = (1,)`
		`print(t)`
		`OUTPUT: (1,)`

Sets

- Sets are unique collections of elements
- We represent a set using curly braces { }
- Sets can store strings, tuples, and booleans inside them at once.
- Sets cannot store Lists, Sets, and dictionaries within them.
- We cannot index elements from a set because they are not ordered.

Description	Syntax	Example

Declaring a set	`set_name = {elem1, elem2,...}`	`s = {1, 2, 2, 3, 4}`
		`print(s)`
		`OUTPUT: {1, 2, 3, 4}`

Declaring an empty set	`set_name = set()`	`s = set()`
		`print(s)`
		`OUTPUT: set()`

Set Methods

Description	Syntax	Example

• **Adding** an **element** to a set	`set_name.add(element)`	`s = {1, 2, 2, 3, 4}`
		`print(s.add(5))`
		`OUTPUT: {1, 2, 3, 4, 5}`

• **Removing** an **element** from a set.	`set_name.remove(element)`	`s = {1, 2, 2, 3, 4}`
		`print(s.remove(2))`
		`OUTPUT: {1, 3, 5}`

• **Popping** a **random element** from a set.	`set_name.pop()`	`s = {1, 2, 2, 3, 4}`
		`print(s.pop())`
		`OUTPUT: 1`

• **Updating multiple elements** in a set.	`set_name.update(elements_to_update)`	`s = {1}`
		`s.update((2, 3, 4, 4, 5))`
		`print(s)`
		`OUTPUT: {1, 2, 3, 4, 5}`

Set Operations

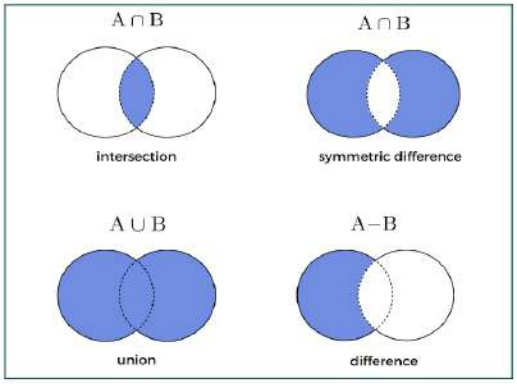
Description	Syntax	Example

Intersection: contains elements common to both sets	`set1.intersection(set2)`
	or
	`set1 & set2`
		`s1 = {2, 3, 4, 5}`
		`s2 = {1, 3, 4, 6, 7, 8}`
		`print(s1 & s2)`
		`OUTPUT: {3, 4}`

Union: contains all elements in both sets	`set1.union(set2)`
	or
	`s1 | s2`
		`s1 = {2, 3, 4, 5}`
		`s2 = {1, 3, 4, 6, 7, 8}`
		`print(s1 | s2)`
		`OUTPUT: {1, 2, 3, 4, 5, 6, 7, 8}`

Difference: contains all the elements in set1 which are not in set2	`set1.difference(set2)`
	or
	`s1 - s2`
		`s1 = {2, 3, 4, 5}`
		`s2 = {1, 3, 4, 6, 7, 8}`
		`print(s1 - s2)`
		`OUTPUT: {2, 5}`

Symmetric difference: contains all the elements in union minus the common elements	`set1.symmetric_difference(set2)`
	or
	`s1 ^ s2`
		`s1 = {2, 3, 4, 5}`
		`s2 = {1, 3, 4, 6, 7, 8}`
		`print(s1 ^ s2)`
		`OUTPUT: {1, 2, 5, 6, 7, 8}`



Dictionaries

Dictionaries are key-value pairs

Description	Syntax	Example
Creating dictionaries in python	`dict_name = {key1: value1, key2: value2, ...}`	`d = {"delhi": 450, "up": 700}`

Accessing a value from a dictionary	`dict_name[key_to_access]`	`d = {"delhi": 450, "up": 700}`
		`print(d["up"])`
		`OUTPUT: 700`

Dict.get() used to **access** a **value** from a dictionary. advantage: does not throw an error if the key is not present in the dictionary.	`dict_name.get(key_to_access)`	`d = {"delhi": 450, "up": 700}`
		`print(d.get("up"))`
		`OUTPUT: 700`
		`print(d.get("haryana"))`
		`OUTPUT: NONE`

Updating a **value** in a dictionary.	`dict_name[key] = new_value`	`d = {"delhi": 450, "up": 700}`
		`d["delhi"] = 500`
		`print(d)`
		`OUTPUT: {"DELHI": 500, "UP": 700}`

Removing a **key** from the dictionary.	`dict_name.pop(key_to_pop)`	`d = {"delhi": 450, "up": 700}`
		`d.pop("up")`
		`print(d)`
		`OUTPUT: {"DELHI": 450}`