FUNDAMENTAL PYTHON TERMS & CONCEPTS

TERM / CONCEPT	EXPLANATION	EXAMPLE	
Syntax	The “grammar” of Python — rules for how code must be written so the computer understands it.	print("Hello") must have parentheses, quotes, and correct indentation.	
Indentation	The spaces at the beginning of a line that show which code belongs together.	python\nif True:\n print("Indented!")\n	
Variable	A name that stores a value (like a box with a label).	x = 10 or name = "Nathaniel"	
Data Type	The kind of data stored (like number, text, list, etc.).	int, float, str, list, dict, bool	
String (str)	Text inside quotes.	"Hello" or 'World'	
Integer (int)	Whole number (no decimals).	10, -5, 100	
Float (float)	Decimal number (has a point).	3.14, -2.5	
Boolean (bool)	True or False value.	True, False	
List	A collection of items, ordered and changeable.	[1, 2, 3], ["a", "b", "c"]	
Tuple	Ordered collection, but cannot be changed.	(1, 2, 3)	
Set	Unordered collection, no duplicates allowed.	{1, 2, 3}	
Dictionary (dict)	Stores data in key–value pairs.	{"name": "Nathaniel", "age": 21}	
Index	The position number of an item in a list or string (starts at 0).	"Hello"[0] → 'H'	
Slice	A section or range of items.	numbers[1:4] → items 1, 2, 3	
Function	A block of code that does one job and can be reused.	python\ndef greet():\n print("Hi!")\n	
Argument	Data you give a function to use.	print("Hi") → "Hi" is the argument.	
Return	Sends a value back from a function.	python\ndef add(x, y):\n return x + y\n	

Module / Library	Pre-written code you can import to use.	import random, import math	
Import	Tells Python to bring in a module.	import random	
Loop	Repeats code multiple times.	for i in range(5): print(i)	
For Loop	Repeats for each item in something.	for x in [1,2,3]: print(x)	
While Loop	Repeats while a condition is true.	while x < 10: x += 1	
Break	Stops a loop early.	if x == 5: break	
Continue	Skips to the next loop cycle.	if x == 2: continue	
Pass	Placeholder; does nothing but avoids an error.	python\ndef future_function(): pass	
Condition	A statement that can be True or False.	if age > 18:	
If / Elif / Else	Used to make decisions in code.	python\nif x > 10:\n print("Big")\nelif x == 10:\n print ("Equal")\nelse:\n print("Small")\n	
Operator	A symbol that does math or logic.	"=+, -, ==, and, or"	
Comment	Notes in your code (Python ignores them).	# This is a comment	
Input / Output (I/O)	Getting data from user (input) and showing it (output).	input(), print()	
Type Casting	Converting one data type to another.	int("5"), str(10)	
Error / Exception	What happens when Python hits a problem.	ZeroDivisionError, TypeError	
Try / Except	Used to catch and handle errors safely.	python\ntry:\n 1/0\nexcept ZeroDivisionError:\n print("Can't divide by zero!")\n	
Class	Blueprint for creating objects (used in OOP).	python\nclass Dog:\n def __init__(self, name):\n self.name = name\n	
Object	An instance of a class (a created “thing”).	dog1 = Dog("Buddy")	
Attribute	A variable that belongs to an object.	dog1.name	
Method	A function inside a class.	dog1.bark()	

Scope	Where in the code a variable can be used.	Variables inside a function only work inside it.	
Global Variable	A variable usable anywhere in the file.	global_var = 10	
Local Variable	A variable usable only inside its function.	Defined inside def.	
Comment Docstring	Multi-line explanation for a function/class.	"""This function adds two numbers."""	
None	A special value meaning “nothing” or “empty.”	x = None	
len()	Returns the number of items or characters.	len("Python") → 6	

MOST COMMON PYTHON KEYWORDS & FUNCTIONS

KEYWORD / FUNCTION	EXPLANATION	EXAMPLE	OUTPUT / NOTES
print()	Shows text or numbers on the screen.		
if	Checks a condition, and runs code if it’s true.		
elif	Means “else if.” Another condition to check if the first one wasn’t true.		
else	Runs code if none of the if or elif conditions were true.		
while	Repeats code while a condition is true.		
for	Loops over a sequence (like a list or a range of numbers).		
break	Stops the loop completely, even if it’s not finished.		
continue	Skips the rest of the loop for this round, and goes to the next.		
pass	Does nothing (a placeholder so code doesn’t break).		
def	Used to create a function (a mini machine).		

return	Sends a value back from a function.		
Import	Brings in extra tools (modules/libraries).		
from ... import ...	Brings in a specific tool from a module.		
class	Creates a blueprint for objects (used in OOP).		
with	Sets up a temporary block of code (often for files).		
try	Tests code that might cause an error.		
except	Runs if an error happens.		
finally	Always runs after try/except, error or not.		
TRUE	A value meaning “yes/it’s correct.”		
FALSE	A value meaning “no/it’s not correct.”		
None	A special value meaning “nothing here.”		
and	Checks if both conditions are true.		
or	Checks if at least one condition is true.		
not	Flips a condition (True → False, False → True).		
in	Checks if something is inside a list, string, or range.		
is	Checks if two things are the exact same object in memory.		
len()	Finds how many items are in a list, string, etc.		
range()	Creates a sequence of numbers (used in loops).		
input()	Lets the user type something in.		
int()	Turns something into a whole number.		

float()	Turns something into a decimal number.		
str()	Turns something into text.		
list()	Turns something into a list.		
dict()	Creates a dictionary (key-value storage).		
set()	Creates a set (a collection with no duplicates).		

LISTS & TUPLES

TOPIC / KEYWORD	EXPLANATION	EXAMPLE	OUTPUT / NOTES
len()	Returns the number of items.	len([1, 2, 3]) → 3	List / Tuple
[] (Indexing)	Access specific item by position.	[1, 2, 3][0] → 1	List / Tuple
[:] (Slicing)	Returns a part of the list/tuple.	[1, 2, 3][0:2] → [1, 2]	List / Tuple
.append()	Adds an item to the end.	list.append(4) → [1, 2, 3, 4]	List
.insert()	Adds an item at a specific position.	list.insert(1, 9) → [1, 9, 2, 3]	List
.remove()	Removes first matching value.	[1, 2, 2, 3].remove(2) → [1, 2, 3]	List
.pop()	Removes item at index (default last) and returns it. You can assign the popped value to a new variable!! super useful	[1, 2, 3].pop() → 3	List ***I love this method***
.clear()	Removes all items.	[1, 2, 3].clear() → []	List
.index()	Returns index of first matching value.	[1, 2, 3].index(2) → 1	List
.count()	Counts occurrences of a value.	[1, 2, 2, 3].count(2) → 2	List
.sort()	Sorts items ascending.	[3, 1, 2].sort() → [1, 2, 3]	List

.reverse()	Reverses item order.	[1, 2, 3].reverse() → [3, 2, 1]	List
.copy()	Returns a copy of the list.	new_list = list.copy()	List
tuple()	Converts a list into a tuple.	tuple([1, 2, 3]) → (1, 2, 3)	List → Tuple
list()	Converts a tuple into a list.	list((1, 2, 3)) → [1, 2, 3]	Tuple → List
in	Checks if an item exists. Returns True/False.	2 in [1, 2, 3] → True	List / Tuple
is	Checks if two objects are the same in memory.	[1] is [1] → False	List / Tuple
Immutable property	Tuples can’t be changed after creation.	(1, 2)[0] = 9 → Error	Tuple
Mutable property	Lists can be changed after creation.	[1, 2][0] = 9 → Works	List

DICTIONARIES, FUNCTIONS, AND MATRICES

TOPIC / KEYWORD	EXPLANATION	EXAMPLE	OUTPUT / NOTES
Dictionary	A collection of data stored in key: value pairs.	person = {"name": "Nathaniel", "age": 21}	{"name": "Nathaniel", "age": 21}
dict()	Creates a dictionary using the dict function.	person = dict(name="Nathaniel", age=21)	Same as above
Access value	Get a value using its key.	person["name"]	"Nathaniel"
Add / Change item	Add a new key or change an existing value.	person["city"] = "Abilene"	Adds new key-value pair
Remove item	Remove a key-value pair.	person.pop("age")	Removes "age" key
.keys()	Returns all the keys.	person.keys()	dict_keys(['name', 'city'])
.values()	Returns all the values.	person.values()	dict_values(['Nathaniel', 'Abilene'])
.items()	Returns all key-value pairs as tuples.	person.items()	[('name', 'Nathaniel'), ('city', 'Abilene')]

.get()	Safely get a value (no error if missing).	person.get("age", "Not found")	"Not found"
.update()	Adds or updates multiple items at once.	person.update({"age": 21, "job": "Developer"})	Adds both keys
.clear()	Removes everything from the dictionary.	person.clear()	{}
del	Deletes a key or the whole dictionary.	del person["name"]	Removes key
Loop through dictionary	Go through each key in the dictionary.	for key in person: print(key, person[key])	Prints each key/value
Check if key exists	See if a key is in the dictionary.	"name" in person	TRUE

FUNCTIONS (USING DEF)

TOPIC / KEYWORD	EXPLANATION	EXAMPLE	OUTPUT / NOTES
def	Defines (creates) a function.	def greet(): print("Hello")	Creates a function
Call function	Runs the code inside the function.	greet()	Prints “Hello”
Arguments	Inputs you give to the function.	def greet(name): print("Hi", name)	Needs a name to run
Call with argument	Give the function its input.	greet("Nathaniel")	Prints “Hi Nathaniel”
Return	Sends a value back from a function.	def add(a, b): return a + b	add(2, 3) → 5
Default argument	Gives an argument a default value.	def greet(name="friend"): print("Hi", name)	Works even if you don’t give a name
Multiple arguments	Use commas to separate them.	def multiply(a, b): return a * b	multiply(2, 4) → 8
*args	Lets you pass any number of arguments.	def total(*nums): return sum(nums)	total(1,2,3) → 6
**kwargs	Lets you pass key=value pairs.	def show(**info): print(info)	show(name="Nate", age=21) → {'name':'Nate','age':21}

MATRICES (LISTS OF LISTS)

CONCEPT	EXPLANATION	EXAMPLE	OUTPUT / NOTES
Matrix	A list that holds lists (like rows and columns).	matrix = [[1,2,3], [4,5,6], [7,8,9]]	3x3 grid of numbers
Access element	Use two indexes: [row][column].	matrix[0][1]	2 (1st row, 2nd column)
Change value	Reassign an element.	matrix[1][1] = 99	Changes middle value
Loop through matrix	Use nested loops.	for row in matrix: print(row)	Prints each row
Flatten matrix	Turn all values into one list.	[num for row in matrix for num in row]	[1,2,3,4,5,6,7,8,9]
Matrix length	Get number of rows or columns.	len(matrix)	3 rows
2D comprehension	Quick way to build a matrix.	[[x*y for x in range(3)] for y in range(3)]	3x3 multiplication table

MATHEMATICAL & COMPARISON OPERATORS & LOGICAL OPERATORS (USED FOR CONDITIONS)

SYMBOL / OPERATOR	EXPLANATION	EXAMPLE CODE	OUTPUT / NOTES
"+" Addition	Adds numbers together	3 + 2	5
- Subtraction	Subtracts right from left	7 - 4	3
* Multiplication	Multiplies numbers	5 * 3	15
/ Division	Divides left by right (always gives a float)	10 / 2	5
// Floor Dvision	Divides and rounds down to whole number	10 // 3	3
% Modulus (Remainder)	Gives the remainder after division	10 % 3	1

** Exponent	Raises to a power (2³)	2 ** 3	8
"=" Assignment	Stores 5 inside x	x = 5	x is 5
"=+' Add & Assign	Same as x = x + 2	x += 2	x increases by 2
-= Subtract & Assign	Same as x = x - 1	x -= 1	x decreases by 1
*= Multiply & Assign	Same as x = x * 3	x *= 3	x triples
/= Divide & Assign	Same as x = x / 2	x /= 2	x is halved
**= Power & Assign	Squares (or powers) x	x **= 2	
%= Mod & Assign	Stores remainder of x / 3	x %= 3	
"==" Equal to	Checks if both sides are equal	5 == 5	TRUE
!= Not equal to	Checks if sides are different	5 != 3	TRUE
> Greater than	True if left is bigger	7 > 3	TRUE
< Less than	True if left is smaller	2 < 5	TRUE
>= Greater than or equal	True if left ≥ right	5 >= 5	TRUE
<= Less than or equal	True if left ≤ right	4 <= 7	TRUE
and Logical AND	True if both are true	(5 > 2) and (4 < 10)	TRUE
or Logical OR	True if either is true	(5 > 10) or (3 < 5)	TRUE
not Logical NOT	Flips True ↔ False	not (5 > 2)	FALSE

BEST PRACTICES / STYLE GUIDE RULES FROM PEP 8 (THE OFFICIAL PYTHON STYLE GUIDE)

CATEGORY	BEST PRACTICE /	WHY / WHAT IT	EXAMPLE /
Indentation	Use 4 spaces per indent level, never tabs.	Consistency makes code blocks clear. (Python Enhancement Proposals (PEPs))	
Maximum Line Length	Aim for ~79 characters per line (comments ~72).	Helps in readability, side-by-side viewing. (Real Python)	
Blank Lines	Use blank lines to separate top-level function/class definitions (2 lines) and methods inside classes (1	Helps visually separate logical sections. (Real Python)	
Import	- Each import on its own line.- Imports go at top of file, after module comments/docstrings.- Group	Keeps dependencies organized and readable. (Python Enhancement Proposals (PEPs))	
Whitespace / Spacing	- Use spaces around operators and after commas.- Don't put spaces inside parentheses, brackets, or	Makes expressions easier to parse visually. (pythoncentral.io)	Example: x = (a + b) * c not x=(a +b)* c.
Naming Conventions	- Functions, variables, modules: lowercase_with_underscores.- Classes: CapWords	Makes code more "Pythonic" and predictable. (Python Enhancement Proposals (PEPs))	
Comments & Docstrings	- Use complete sentences starting with a capital letter.- Update comments when code changes.- Use	Helps others (and future you) understand intent. (pythoncentral.io)	
Quotes for Strings	Either single ' or double " quotes are okay — be consistent.	Avoids unnecessary escaping. (pythoncentral.io)	If you have an apostrophe inside, using double quotes may avoid escapes: "John's book"
Line Continuation / Wrapping	- Prefer implicit continuation inside (), [], {}.- If you break a line, do so before a binary operator rather	Keeps readability when lines get long. (Real Python)	
Avoid Excessive Alignment	Don't align variable assignments across many lines just for looks (unless it improves clarity).	Over-alignment can make code rigid or harder to change. (Discussions on Python.org)	Instead of x = 1 / long = 2, do x = 1 / long = 2.
Don't Use Mutable Default Arguments	Avoid using lists, dictionaries, etc. as default arguments in function definitions.	They can persist between calls and cause bugs.	def func(my_list=None): if my_list is None: my_list = [] …
One Statement per Line	Keep to one logical statement per line (avoid chaining multiple statements with ;).	Clarity, easier debugging.	
Truth Comparisons	Use is None or is not None rather than == None. Use if x: rather than if x == True:.	Cleaner, more Pythonic.	
Exception Handling	Use specific exception types rather than a bare except:.	Avoids catching unintended errors.	
Consistent Style	Be consistent within a project or module.	Slight deviations are okay if they improve readability in that context. (Python Enhancement	

SUPER USEFUL BUILT-IN FUNCTIONS (*NOT EMPHASIZED IN PCEP*)

FUNCTION	WHAT IT DOES	EXAMPLE CODE	WHY IT'S USEFUL
sum()	Adds all numbers in a list, tuple, etc.	sum([10, 20, 30])	Replaces manual for loops for totals.
max()	Finds the largest value.	max([3, 7, 2])	Perfect for quick comparisons.
min()	Finds the smallest value.	min([3, 7, 2])	Great for finding limits.
sorted()	Returns a new sorted list.	sorted([3, 1, 2])	Sorts without changing original data.
reversed()	Returns the sequence in reverse order.	list(reversed([1, 2, 3]))	Handy for going backward through lists.
any()	Returns True if any value is True.	any([False, True, False])	Quick way to check multiple conditions.
all()	Returns True only if all are True.	all([True, True, False])	Great for validation checks.
zip()	Combines multiple lists into pairs.	zip([1, 2, 3], ['a', 'b', 'c'])	Clean way to loop through several lists together.
enumerate()	Adds index numbers automatically.	for i, val in enumerate(['a','b']): print(i, val)	Replaces manual counters in loops.
map()	Applies a function to every item in a list.	map(str, [1, 2, 3]) → ['1','2','3']	Replaces repetitive for loops.
filter()	Keeps only values that meet a condition.	filter(lambda x: x > 5, [2, 8, 3]) → [8]	Great for cleaning data.
round()	Rounds to 2 decimal places.	round(3.14159, 2)	Common for money or measurements.
abs()	Returns the absolute (positive) value.	abs(-5) → 5	Used often in math-heavy code.
divmod()	Gives quotient & remainder at once.	divmod(10, 3) → (3, 1)	Replaces // and % together.
set()	Removes duplicates automatically.	set([1, 2, 2, 3]) → {1, 2, 3}	Cleans up lists easily.
zip(*iterables)	Reverses easily: zip(*pairs) → splits lists again	zip([1,2],[3,4]) → pairs	Used in data manipulation.

PYTHON EXCEPTIONS & HOW TO USE THEM

CONCEPT / KEYWORD	MEANING /	EXAMPLE CODE	EXPLANATION
try:	Marks a block of code to attempt. Python will “try” to r	python try: print(10 / 0)	Starts a “watch zone” for possible errors.
except:	Catches and handles the error if one occurs.	python except: print("Something went wrong!")	Prevents crash; runs this instead.
except ExceptionName:	Catches a specific kind of error.	python try: print(10 / 0) except ZeroDivisionError: pri	Only catches division-by-zero errors.
else:	Runs if no error happens inside the try block.	python try: print("Hi") except: print("Error") else: prin	“Success” code — runs when all goes well.
finally:	Always runs, error or not. Often used for cleanup.	python try: print(1 / 0) except: print("Error") finally: pr	Useful for closing files, connections, etc.
raise	Manually triggers an error.	python raise ValueError("Bad input!")	Lets you create intentional, custom errors.
assert	Tests a condition; raises AssertionError if false.	python assert 2 + 2 == 4	Used for debugging and sanity checks.
Exception	The base class for all exceptions.	python except Exception as e: print(e)	Catches any type of error.
as	Lets you name the exception for details.	python try: 1/0 except ZeroDivisionError as err: print	Gives you access to the actual error message.

RANDOM MODULE – KEY FUNCTIONS

FUNCTION	WHAT IT DOES	NOTES	
random.random()	Returns a random float in the range [0.0, 1.0). (Python documentation)	The random module is huge — especially with distribution functions — so this list covers the more commonly used ones for general coding and beginner/intermediate use.	
random.uniform(a, b)	Returns a random float between a and b (inclusive of endpoints in concept) (Python documentation)		
random.randint(a, b)	Returns a random integer N such that a ≤ N ≤ b. (You've used this) (GeeksforGeeks)		
random.randrange(start, stop [, step])	Returns a randomly selected element from range (start, stop, step) (W3Schools)		
random.choice(seq)	Returns a random element from the non-empty sequence seq. (GeeksforGeeks)		
random.choices(population, weights=None, *, cum_weights=None, k=1)	Returns a list of k selections with replacement, optionally weighted. (Python documentation)		
random.sample(population, k)	Returns k unique elements chosen from the population (no repeats) (GeeksforGeeks)		
random.shuffle(x)	Shuffles the sequence x in place. (GeeksforGeeks)		
random.getrandbits(k)	Returns a random integer with k random bits. (W3Schools)		
random.seed(a=None, version=2)	Initializes the random number generator for reproducible results. (GeeksforGeeks)		
random.getstate()	Returns the current internal state of the generator. (W3Schools)		
random.setstate(state)	Restores the internal state of the generator to the supplied state object. (W3Schools)		
Real-valued distribution functions (a few)	For example: betavariate(alpha, beta), expovariate (lambd), gammavariate(alpha, beta), gauss(mu, sigma), lognormvariate(mu, sigma), vonmisesvariate (mu, kappa), paretovariate(alpha), weibullvariate (alpha, beta). (W3Schools)		
random.choice()	Selects a random value/item from lists/tuples/dictionaries. ***This works for strings and numbers***		

STATISTICS MODULE – KEY FUNCTIONS

FUNCTION	WHAT IT DOES	NOTES	
statistics.mean(data)	Returns the arithmetic mean ("average") of the data. (Python documentation)	The statistics module is lighter than random in terms of number of functions, but very useful when you're working with numeric data, analysis, or reporting.	
statistics.median(data)	Returns the median (middle value) of the data. (What you used) (Python documentation)		
statistics.median_low(data)	Returns the low median (for even-length data, returns the lower of the two middle values) (Python documentation)		
statistics.median_high(data)	Returns the high median (for even-length data, returns the higher of the two middle values) (Python documentation)		
statistics.median_grouped (data, interval=1)	For data grouped in intervals, returns an estimate of median. (Python documentation)		
statistics.mode(data)	Returns the single most common data point. (TutorialsTeacher)		
statistics.multimode(data)	Returns a list of the most common values (in case of ties). (Python documentation)		
statistics.pvariance(data)	Population variance of the data. (Python documentation)		
statistics.variance(data)	Sample variance (uses n-1 denominator) (Python documentation)		
statistics.pstdev(data)	Population standard deviation of the data. (Python documentation)		
statistics.stdev(data)	Sample standard deviation of the data. (Python documentation)		
statistics.quantiles(data, n=4, method='exclusive')	Returns n − 1 cut points dividing the data into n equal-sized subsets. (Python documentation)		

PYTHON OS MODULE – SELECTED USEFUL FUNCTIONS

FUNCTION	WHAT IT DOES	NOTES	
os.name	Gives the name of the operating system dependent module (like "posix" or "nt").	used alongside SHUTIL module >>>	
os.getcwd()	Returns the current working directory (where your script is running).		
os.chdir(path)	Changes the current working directory to the given path.		
os.listdir(path='.')	Returns a list of entries (files + directories) in the given directory (defaults to current).		
os.mkdir(path[, mode])	Creates a new directory at the specified path.		
os.makedirs(path[, mode, exist_ok])	Creates all intermediate-level directories needed for the specified path.		
os.rmdir(path)	Removes (deletes) an empty directory at the given path.		
os.remove(path)	Deletes the file at the specified path.		
os.rename(src, dst)	Renames a file or directory from src to dst.		
os.renames(old, new)	Recursively renames/moves directories and files—moves entire subtree if needed.		
os.stat(path)	Retrieves information about the file or directory at path (size, modification time, permissions, etc.).		
os.access(path, mode)	Checks if path can be accessed with the given mode (read/write/execute).		
os.environ	A mapping object (like a dictionary) representing the environment variables of the system.		
os.getenv(key[, default])	Retrieves the value of the environment variable named key, or default if not present.		
os.system(command)	Executes the command (string) in the system shell; returns exit status.		
os.walk(top[, topdown=True, onerror=None, followlinks=False])	Generator that yields directory tree: for each directory, gives path, directories list, files list — great for exploring folders recursively.		

SHUTIL MODULE — QUICK REFERENCE TABLE

FUNCTION	PURPOSE	EXAMPLE	NOTES
shutil.move(src, dest)	Move a file or folder	shutil.move("file.txt", "Folder/")	Actually relocates the item
shutil.copy(src, dest)	Copy a file (no metadata)	shutil.copy("a.txt", "b.txt")	Creates a new file
shutil.copy2(src, dest)	Copy with metadata	shutil.copy2("a.txt", "Backup/")	Preserves dates/permissions
shutil.copytree(src, dest)	Copy an entire folder	shutil.copytree("Old", "New")	Recursive — copies everything
shutil.rmtree(path)	Delete a folder + contents	shutil.rmtree("Temp/")	Dangerous — no undo
shutil.disk_usage(path)	Check drive space	shutil.disk_usage("/")	Returns total, used, free
shutil.which(cmd)	Locate an executable	shutil.which("python")	Shows the full path
shutil.make_archive(base, form	Create a zip/tar archive	shutil.make_archive("backup", "zip", "Project")	Turns a folder into a zip file
shutil.unpack_archive(filename	Extract zip/tar archive	shutil.unpack_archive("backup.zip")	Works on many formats

PYTHON REQUESTS MODULE — KEY METHODS & USAGE

METHOD	WHAT IT DOES	NOTES	
requests.get(url, params=None, **kwargs)	Sends an HTTP GET request to the specified URL. (w3schools.com)	The **kwargs in those methods let you specify things like headers, cookies, timeout, auth, params, json, data, verify, etc. w3schools.com +1	
requests.post(url, data=None, json=None, **kwargs)	Sends an HTTP POST request to send data to the server (often used for form submissions or APIs). (Real Python)		

requests.put(url, data=None, **kwargs)	Sends an HTTP PUT request (commonly used to update/replace a resource). (GeeksforGeeks)	Using response.json() is a handy way to get JSON payloads returned by many web APIs. Always check response.status_code (e.g., 200 = success) and optionally use response. raise_for_status() to handle non-successful responses. Because requests is third-party (not built into Python standard library), you need to install it with pip install requests. PyPI	
requests.delete(url, **kwargs)	Sends an HTTP DELETE request (retrieve or remove resource). (GeeksforGeeks)		
requests.head(url, **kwargs)	Sends an HTTP HEAD request (like GET but only retrieves headers, no body). (GeeksforGeeks)		
requests.patch(url, data=None, **kwargs)	Sends an HTTP PATCH request (partial update of a resource). (Real Python)		
requests.options(url, **kwargs)	Sends an HTTP OPTIONS request (asks server what methods are allowed/available). (Medium)		
requests.request(method, url, **kwargs)	The flexible general method — you specify the HTTP method as a parameter ("GET", "POST", etc.). (Stack Overflow)		
response = requests.X(...) where X is one of the above — then from the response object you can use: response.status_code, response.text, response.json(), response.headers, etc. (Real Python)			

PYTUBE - MAIN CONCEPTS

CONCEPT	MEANING		
YouTube	The main class used to access a video		
Streams	Every format of a video: resolution, audio, mp4, webm, etc		
Filters	Narrow down which stream you want		
Download()	Saves the file to your computer		

PYTUBE - KEY CLASSES & FUNCTIONS

Function / Class	Usage	Example	
YouTube(url)	Creates a YouTube object from a video link	yt = YouTube("https://youtu.be/...")	

.title	Returns the video title	yt.title	
.author	Video creator	yt.author	
.views	Number of views	yt.views	
.length	Duration in seconds	yt.length	
.thumbnail_url	Thumbnail image URL	yt.thumbnail_url	

PYTUBE - WORKING WITH STREAMS

Method / Attribute	Description	Example	
.streams	All available streams	yt.streams	
.streams.filter()	Filter by quality, audio, etc	yt.streams.filter(res="720p")	
.get_highest_resolution()	Returns the highest quality video stream	yt.streams.get_highest_resolution()	
.get_audio_only()	Audio-only version	yt.streams.get_audio_only()	
.first()	Get the first match after filtering	yt.streams.filter(progressive=True).first()	

PYTUBE - DOWNLOADING

Method	Description	Example	
.download()	Download video or audio	stream.download()	
.download(output_path="path")	Save to a specific folder	stream.download(output_path="Videos/")	
.download(filename="name.mp	Choose file name	stream.download(filename="MyVideo.mp4")	

PYTUBE - PROGRESS CALLBACKS (OPTIONAL)

Feature	Purpose		
on_progress_callback	Track download progress		
on_complete_callback	Trigger after download finishes		

INPUT FUNCTION

TRICK / USE	EXPLANATION	EXAMPLE CODE	
Basic input	Asks the user a question and stores their answer.	name = input("What is your name? ")	
Print and input combo	You can use both to interact with users.	age = input("Enter your age: "); print("You are", age)	
Convert input to number	By default, input() gives text (a string). Use int() or float() for numbers.	num = int(input("Enter a number: "))	
Store multiple inputs	Splits user input into parts — example: typing 3 5 gives a=3, b=5.	a, b = input("Enter two numbers: ").split()	
Convert multiple inputs to integers	Converts both to numbers at once.	a, b = map(int, input("Enter two numbers: ").split())	
Use default value if empty	If user presses Enter, it uses “Guest.”	name = input("Name (default=Guest): ") or "Guest"	
Add symbols or emojis in prompt	You can use any text or emoji in your prompt.	input(" Enter a number: ")	
Hide user input (password)	Hides what the user types (for passwords).	python\nimport getpass\npassword = getpass.getpass("Enter password: ")\n	
Use f-string with input	Lets you use variables inside the prompt.	item = input(f"What would you like to buy, {name}? ")	
Limit length manually	You can check and control how long input is.	python\nuser = input("Username: ")\nif len(user) > 10:\n print("Too long!")\n	
Loop until valid input	Keeps asking until user types a valid number.	python\nwhile True:\n age = input("Enter age: ")\n if age.isdigit(): break\n	
Use .strip() to clean spaces	Removes extra spaces before or after input.	name = input("Name: ").strip()	
Capitalize / Lowercase input	Converts input to lowercase for easier checking.	answer = input("Yes or no: ").lower()	
Quick input test	Whatever user types is printed right back.	print(input("Type something: "))	
Input + calculation	Takes user input, does math, prints result.	python\nnum = int(input("Enter a number: "))\nprint(num * 10)\n	

PRINT FUNCTION

TRICK / USE	EXPLANATION	EXAMPLE CODE	
Basic print	Prints text or numbers to the screen.	print("Hello, world!")	
Print multiple things	You can print many items separated by commas.	print("Age:", 25)	
Add custom separator	Adds a custom separator instead of spaces → 2025-10-07.	print("2025", "10", "07", sep="-")	
End without new line	By default, print ends with a new line. end="" stops that.	print("Loading...", end="")	
End with something else	Makes it print on one line, adding custom text at the end.	print("Step 1", end=" ➜ ")	
Print with escape characters	\n makes a new line. Other examples: \t (tab), \\ (backslash).	print("Line 1\nLine 2")	
Formatted output (f-string)	f-strings let you put variables inside {} easily.	name = "Nathaniel"; print(f"Hello, {name}!")	
Old format method	Older way to insert values into text.	print("Hello, {}".format("Nathaniel"))	
Print quotes inside quotes	Mix ' and " to include quotes inside text.	print('He said "Hi" to me!')	
Combine text and math	You can print results of math operations too.	print("2 + 3 =", 2 + 3)	
Print lists or tuples	Prints whole lists or tuples at once.	nums = [1,2,3]; print(nums)	
Print without spaces between items	Prints letters together → ABC.	print("A","B","C", sep="")	
Flush output (advanced)	Forces text to appear immediately (used in live output situations).	print("Hello", flush=True)	
Multiline print with triple quotes	Triple quotes let you write long multi-line text easily.	print("""Line 1\nLine 2""")	
Use emojis / symbols	Python can print emojis and special symbols.	print(" Task done!")	
Align text (f-string)	Aligns text left or right — great for tables.	print(f"{'Left':<10}{'Right':>10}")	
Round numbers when printing	Prints only 2 decimals → Value: 3.14.	print(f"Value: {3.14159:.2f}")	

Print variable names and values (Python 3.8+)	Prints both name and value → x=5.	x = 5; print(f"{x=}")	

STRINGS TIPS & TRICKS

CONCEPT / METHOD	EXPLANATION	EXAMPLE CODE	CODE RESULTS	NOTES
Create a string	Stores text inside quotes	name = "Nathaniel"	"Nathaniel"	
Single quotes also work	You can use ' or "	word = 'Hello'	Hello'	
Triple quotes for multi-line	Lets you write text on multiple lines	text = """Line1\nLine2"""	"Line1\nLine2"	
String concatenation	Joins text together	"Hello " + "World"	"Hello World"	
String repetition	Repeats the string 3 times	"Hi" * 3	"HiHiHi"	
Indexing (get one letter)	Gets the first character (count starts at 0)	"Python"[0]	P'	
Negative index	Gets last character	"Python"[-1]	n'	
Slicing	Gets letters from 0 to 2	"Python"[0:3]	Pyt'	
Length of string	Counts how many characters	len("Hello")	5	
Check if substring in string	Checks if text exists inside	"Py" in "Python"	TRUE	
Not in string	Checks if text is missing	"Java" not in "Python"	TRUE	
Uppercase	Makes all letters uppercase	"hello".upper()	"HELLO"	
Lowercase	Makes all letters lowercase	"HELLO".lower()	"hello"	
Capitalize first letter	Only first letter uppercase	"python".capitalize()	"Python"	
Title case	Capitalizes first letter of each word	"hello world".title()	"Hello World"	
Swap case	Flips upper/lower	"HeLLo".swapcase()	"hEllO"	
Count occurrences	Counts how many times a letter appears	"banana".count("a")	3	
Find index	Finds first position of letter	"banana".find("n")	2	
Replace text	Swaps parts of text	"I like cats".replace("cats", "dogs")	"I like dogs"	
Split text	Splits string into list	"a,b,c".split(",")	['a', 'b', 'c']	
Join list into string	Joins list back into one string	",".join(['a','b','c'])	"a,b,c"	
Strip spaces	Removes extra spaces front/back	" hi ".strip()	"hi"	

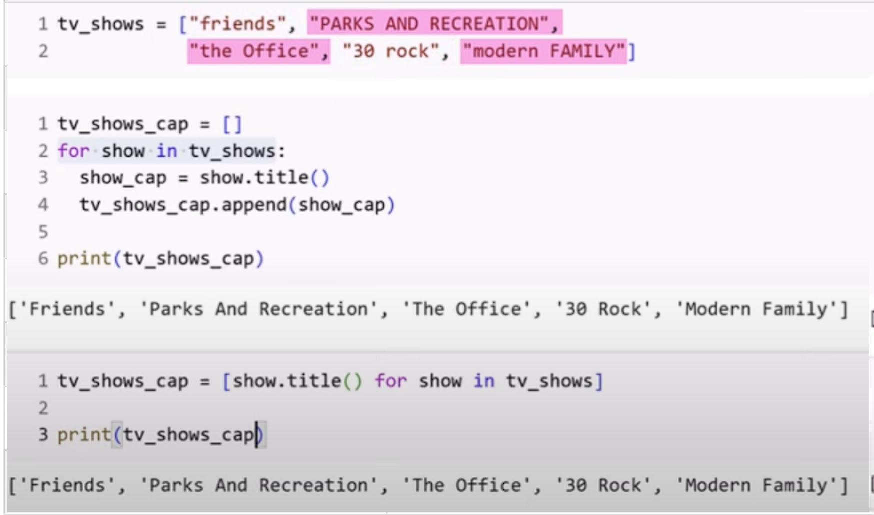
Left strip	Removes only left spaces	" hi".lstrip()	"hi"	
Right strip	Removes only right spaces	"hi ".rstrip()	"hi"	
Starts with	Checks beginning of string	"hello".startswith("he")	TRUE	
Ends with	Checks ending of string	"hello".endswith("lo")	TRUE	
Is numeric	True if all are numbers	"123".isdigit()	TRUE	
Is alphabetic	True if all letters	"abc".isalpha()	TRUE	
Is alphanumeric	True if letters & numbers only	"abc123".isalnum()	TRUE	
Is lower	True if all lowercase	"abc".islower()	TRUE	
Is upper	True if all uppercase	"ABC".isupper()	TRUE	
Center text	Puts text in middle with padding	"Hi".center(10, "-")	"----Hi----"	
Align left	Adds dots to the right	"Hi".ljust(10, ".")	"Hi........"	
Align right	Adds dots to the left	"Hi".rjust(10, ".")	"........Hi"	
Format variables	Inserts variables into strings	f"My name is {name}"	"My name is Nathaniel"	

		HELPFUL VIDEOS	
		LINK / VIDEO	LINK / VIDEO
		Every Python Library / Module Explained in 13 Minutes	(54) 20 Programming Projects That Will Make You A God At Coding - YouTube
INFO	NOTES	django == integrate python with HTML	**Begginer** 1. Porfolio (2:03) 3. To Do List (3:32) 7. Calculator (5:54) 11. Random Quote Generator (9:32) 14. Quiz Program (11:18) 18. Chat Bot (13:15) 20. QR Code Generator (14:03) **Intermediate** 5. Smart Mirror (4:47) 6. Personal Finance Tracker (5:19) 9. Real-time Chat App (7:15) 13. Travel Booking System (10:39) 19. Video Game (13:39) **Advanced** 4. AI Girlfriend/Boyfriend (4:03) 8. Neural Network (6:30) 12. Algorithm Visualizer (10:03) 16. HTTP Server (12:19) 17. Real-time Editor (12:45) **X10 Developer** 2. Build Your Own Git (2:43) 10. Build Your Own Redis (7:53) 15. Build Your Own BitTorrent (11:39)
	NOTES	pygame == create 2d games with python (3d games are possible with help of *OPENGL*)	
	NOTES	piglet == library for making cross platform games already built on top of *OPENGL*	
	NOTES	OpenCV == image recognition, tracking, facial recognition, object control, hand tracking, augmented reality (ar), etc.	
	NOTES	pandas == used for Datascience, built on-top-of NUMPY	
	NOTES	Matplotlib == used for *data-visualization* making and customizing graphs, often used with pandas	
	NOTES	Pillow == known for: image editing but can also be used for: data science, web dev, ai	
	NOTES	fastapi == allows you to make your own API (application programming interface) An API allows programs to interact with eachother	
	NOTES	PYQT == create cross-platform GUI with python. also can use *QTDESIGNER* which cuts down on development time.	
	NOTES	tkinter == create Graphical User Interfaces (GUI)	

	NOTES		
	NOTES		

LIST

[List Comprehensions: Visually Explained]



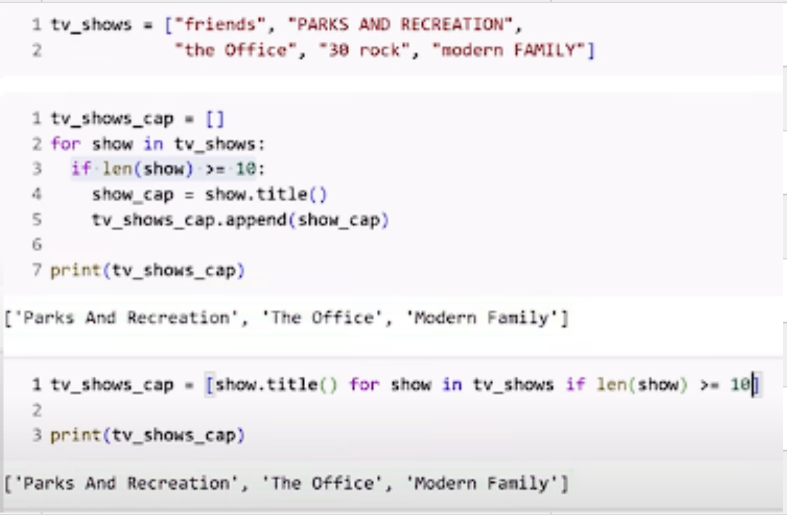
```
tv_shows = ["friends", "PARKS AND RECREATION",
            "the Office", "30 rock", "modern FAMILY"]
```

```
tv_shows_cap = []
for show in tv_shows:
  show_cap = show.title()
  tv_shows_cap.append(show_cap)

print(tv_shows_cap)
```

```
['Friends', 'Parks And Recreation', 'The Office', '30 Rock', 'Modern Family']
```

```
tv_shows_cap = [show.title() for show in tv_shows]

print(tv_shows_cap)
```

```
['Friends', 'Parks And Recreation', 'The Office', '30 Rock', 'Modern Family']
```



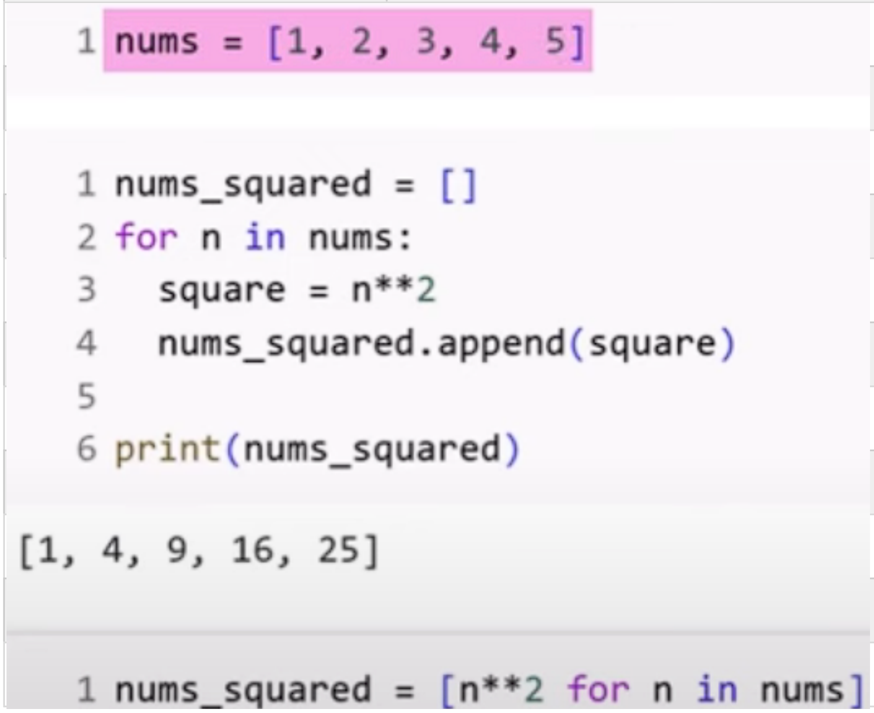
```
tv_shows = ["friends", "PARKS AND RECREATION",
            "the Office", "30 rock", "modern FAMILY"]
```

```
tv_shows_cap = []
for show in tv_shows:
  if len(show) >= 10:
    show_cap = show.title()
    tv_shows_cap.append(show_cap)

print(tv_shows_cap)
```

```
['Parks And Recreation', 'The Office', 'Modern Family']
```

```
tv_shows_cap = [show.title() for show in tv_shows if len(show) >= 10]

print(tv_shows_cap)
```

```
['Parks And Recreation', 'The Office', 'Modern Family']
```

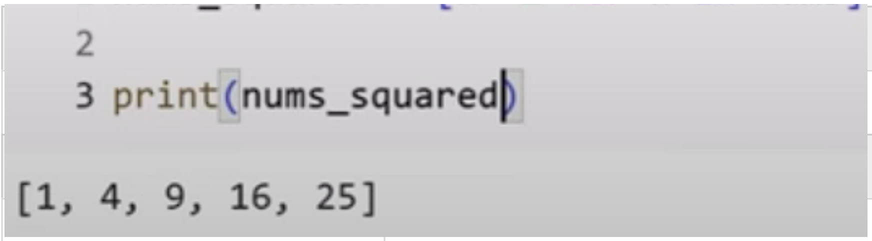


```
nums = [1, 2, 3, 4, 5]
```

```
nums_squared = []
for n in nums:
  square = n**2
  nums_squared.append(square)

print(nums_squared)
```

```
[1, 4, 9, 16, 25]
```

```
nums_squared = [n**2 for n in nums]
```



```

print(nums_squared)
```

```
[1, 4, 9, 16, 25]
```
