1/20/25, 1:16 AM about:blank

Python Programming Fundamentals Cheat Sheet

Package/Method	Description	Syntax and Code Example
AND	Returns 'True' if both statement1 and statement2 are 'True'. Otherwise, returns 'False'.	<pre>Syntax: statement1 and statement2 Example: marks = 90 attendance_percentage = 87 if marks >= 80 and attendance_percentage >= 85: print("qualify for honors") else: print("Not qualified for honors") # Output = qualify for honors</pre>
Class Definition	Defines a blueprint for creating objects and defining their attributes and behaviors.	Syntax: class ClassName: # Class attributes and methods Example: class Person: definit(self, name, age): self.name = name self.age = age
Define Function	A `function` is a reusable block of code that performs a specific task or set of tasks when called.	Syntax: def function_name(parameters): # Function body Example: def greet(name): print("Hello,", name)
Equal(==)	Checks if two values are equal.	Syntax: variable1 == variable2 Example 1: 5 == 5 returns True Example 2: age = 25 age == 30 returns False
For Loop	A `for` loop repeatedly executes a block of code for a specified number of iterations or over a sequence of elements (list, range, string, etc.).	Syntax: for variable in sequence: # Code to repeat Example 1: for num in range(1, 10): print(num) Example 2: fruits = ["apple", "banana", "orange", "grape", "kiwi"] for fruit in fruits: print(fruit)
Function Call	A function call is the act of executing the code within the function using the provided arguments.	Syntax: function_name(arguments) Example: greet("Alice")
Greater Than or Equal To(>=)	Checks if the value of variable1 is greater than or equal to variable2.	Syntax: variable1 >= variable2 Example 1: 5 >= 5 and 9 >= 5 returns True

 	l .	Example 2:
		quantity = 105 minimum = 100 quantity >= minimum
		returns True
		Syntax: variable1 > variable2
Greater Than(>)	Checks if the value of variable1 is greater than variable2.	Example 1: 9 > 6 returns True
		Example 2: age = 20 max_age = 25 age > max_age
		returns False
		Syntax: if condition: #code block for if statement
If Statement	Executes code block `if` the condition is `True`.	Example:
		<pre>if temperature > 30: print("It's a hot day!")</pre>
		Syntax: if condition1: # Code if condition1 is True elif condition2: # Code if condition2 is True else:
If-Elif-Else	Executes the first code block if condition1 is 'True', otherwise checks condition2, and so on. If no condition is 'True', the else block is executed.	# Code if no condition is True Example: score = 85 # Example score if score >= 90:
		print("You got an A!") elif score >= 80: print("You got a B.") else: print("You need to work harder.") # Output = You got a B.
		Syntax: if condition: # Code, if condition is True else: # Code, if condition is False
If-Else Statement	Executes the first code block if the condition is `True`, otherwise the second block.	Example:
		<pre>if age >= 18: print("You're an adult.") else: print("You're not an adult yet.")</pre>
		Syntax:
	Checks if the value of variable1 is less than or equal to variable2.	<pre>variable1 <= variable2 Example 1:</pre>
Less Than or Equal To(<=)		returns True
		Example 2: size = 38 max_size = 40 size <= max_size
Less Than(<)	Checks if the value of variable1 is less than variable2.	returns True Syntax:
Less Thun()	and the same of th	variable1 < variable2
		Example 1:
		4 < 6
	1	· · · · · · · · · · · · · · · · · · ·

		returns True
		Example 2:
		<pre>score = 60 passing_score = 65 score < passing_score</pre>
		returns True
		Syntax:
		for: # Code to repeat
		if # boolean statement break
	'break' exits the loop prematurely. 'continue' skips the rest of the current iteration and moves to the next iteration.	for: # Code to repeat if # boolean statement continue
		Example 1:
Loop Controls		for num in range(1, 6):
		if num == 3: break
		print(num)
		Example 2:
		for num in range(1, 6):
		if num == 3:
		continue print(num)
		Syntax:
		!variable
NOT	Returns 'True' if variable is 'False', and vice versa.	Example:
		!isLocked
		returns True if the variable is False (i.e., unlocked).
	Checks if two values are not equal.	Syntax:
		variable1 != variable2
		Example:
		a = 10 b = 20
Not Equal(!=)		a != b
		returns True
		Example 2:
		count=0
		count != 0
		returns False
		Syntax:
	Creates an instance of a class (object) using the class constructor.	object_name = ClassName(arguments)
Object Creation		Example:
		person1 = Person("Alice", 25)
		ρει 30/11 - Γει 30/11 ΑΙΙΤΕΕ , 23)
OR	Returns 'True' if either statement1 or statement2 (or both) are 'True'. Otherwise, returns 'False'.	Syntax:
		statement1 statement2
		Example:
		"Farewell Party Invitation"
		Grade = 12 grade == 11 or grade == 12
		returns True
range()	Generates a sequence of numbers within a specified range.	Syntax:
		range(stop)
		range(start, stop) range(start, stop, step)
		Example:
		Limitpie.
ı	ı	ı

1/20/25. 1:16 AM about:blank

1/20/25, 1:16 AM about:blank		bout:blank
		<pre>range(5) #generates a sequence of integers from 0 to 4. range(2, 10) #generates a sequence of integers from 2 to 9. range(1, 11, 2) #generates odd integers from 1 to 9.</pre>
Return Statement	'Return' is a keyword used to send a value back from a function to its caller.	Syntax: return value Example: def add(a, b): return a + b result = add(3, 5)
Try-Except Block	Tries to execute the code in the try block. If an exception of the specified type occurs, the code in the except block is executed.	Syntax: try: # Code that might raise an exception except ExceptionType: # Code to handle the exception Example: try: num = int(input("Enter a number: ")) except ValueError: print("Invalid input. Please enter a valid number.")
Try-Except with Else Block	Code in the 'else' block is executed if no exception occurs in the try block.	Syntax: try: # Code that might raise an exception except ExceptionType: # Code to handle the exception else: # Code to execute if no exception occurs Example: try: num = int(input("Enter a number: ")) except ValueError: print("Invalid input. Please enter a valid number") else: print("You entered:", num)
Try-Except with Finally Block	Code in the 'finally' block always executes, regardless of whether an exception occurred.	<pre>Syntax: try: # Code that might raise an exception except ExceptionType: # Code to handle the exception finally: # Code that always executes Example: try: file = open("data.txt", "r") data = file.read() except FileNotFoundError: print("File not found.") finally: file.close()</pre>
While Loop	A `while` loop repeatedly executes a block of code as long as a specified condition remains `True`.	Syntax: while condition: # Code to repeat Example: count = 0 while count < 5: print(count) count += 1



 $\ensuremath{\mathbb{C}}$ IBM Corporation. All rights reserved.

about:blank 4/4