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'.	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

Class Definition	Defines a blueprint for creating objects and defining their attributes and behaviors.	Syntax:
		`class ClassName: # Class attributes and methods`
		Example:
		`class Person:`
		`  def __init__(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 arguements.	Syntax:
		`function_name(arguements)`
		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
		Example 2:
		`quantity = 105`
		`minimum = 100`
		`quantity >= minimum`
		returns True

Greater Than(>)	Checks if the value of variable1 is greater than variable2.	Syntax:
		`variable1 > variable2`
		Example 1: 9 > 6
		returns True
		Example 2:
		`age = 28`
		`max_age = 25`
		`age > max_age`
		returns False

If Statement	Executes code block 'if' the condition is 'True'.	Syntax:
		`if condition: #code block for if statement`
		Example:
		`if temperature > 30:`
		`print("It's a hot day!")`

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.	Syntax:
		`if condition1:`
		# Code if condition1 is True
		`elif condition2:`
		# Code if condition2 is True
		`else:`
		# 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.

If-Else Statement	Executes the first code block if the condition is 'True', otherwise the second block.	Syntax:
		`if condition: # Code, if condition is True`
		`else: # Code, if condition is False`
		Example:
		`if age >= 18:`
		`  print("You're an adult.")`
		`else:`
		`  print("You're not an adult yet.")`

Less Than or Equal To(<=)	Checks if the value of variable1 is less than or equal to variable2.	Syntax:
		`variable1 <= variable2`
		Example 1:
		`5 <= 5 and 3 <= 5`

		returns True Example 2: ``` 1. i 2. 2 3. 3 1. size = 38 2. max_size = 40 3. size <= max_size ``` Copied!
		returns True Syntax: ``` 1. i 1. variable1 < variable2 ``` Copied! Example 1: ``` 1. i 1. 4 < 6 ``` Copied!
Less Than(<)	Checks if the value of variable1 is less than variable2.	returns True Example 2: ``` 1. i 2. 2 3. 3 1. score = 60 2. passing_score = 65 3. score < passing_score ``` Copied!
		returns True Syntax: ``` 1. i 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 1. for: # Code to repeat 2. if # boolean statement 3. break 4. 5. for: # Code to repeat 6. if # boolean statement 7. continue ``` Copied!
Loop Controls	'break' exits the loop prematurely. 'continue' skips the rest of the current iteration and moves to the next iteration.	Example 1: ``` 1. i 2. 2 3. 3 4. 4 1. for num in range(1, 6): 2. if num == 3: 3. break 4. print(num) ``` Copied!
		Example 2: ``` 1. i 2. 2 3. 3 4. 4 1. for num in range(1, 6): 2. if num == 3: 3. continue 4. print(num) ``` Copied!
		Syntax: ``` 1. i 1. !variable ``` Copied!
NOT	Returns 'True' if variable is 'False', and vice versa.	Example: ``` 1. i 1. !isLocked ``` Copied!
		returns True if the variable is False (i.e., unlocked) Syntax: ``` 1. i 1. variable1 != variable2 ``` Copied!
Not Equal(!=)	Checks if two values are not equal.	Example: ``` 1. i 2. 2 3. 3 1. a = 10 2. b = 20 3. a != b ``` Copied!
		returns True Example 2: ``` 1. i 2. 2 1. count=0 2. count != 0 ``` Copied!
		returns False Syntax: ``` 1. i 1. object_name = ClassName(arguments) ``` Copied!
Object Creation	Creates an instance of a class (object) using the class constructor.	Example: ``` 1. i 1. person1 = Person("Alice", 25) ``` Copied!
		Syntax: ``` 1. i 1. statement1 || statement2 ``` Copied!
OR	Returns 'True' if either statement1 or statement2 (or both) are 'True'. Otherwise, returns 'False'.	Example: ``` 1. i 2. 2 1. 'Firewall Party Invitation' 2. Grade = 12 grade == 11 or grade == 12 ``` Copied!
		returns True Syntax: ``` 1. i 2. 2 3. 3 1. range(stop) 2. range(start, stop) 3. range(start, stop, step) ``` Copied!
range()	Generates a sequence of numbers within a specified range.	Example: ``` 1. i 2. 2 3. 3 1. range(5) #generates a sequence of integers from 0 to 4. 2. range(2, 10) #generates a sequence of integers from 2 to 9. 3. range(1, 11, 2) #generates odd integers from 1 to 9. ``` Copied!
		Syntax: ``` 1. i 1. return value ``` Copied!
Return Statement	'Return' is a keyword used to send a value back from a function to its caller.	Example: ``` 1. i 2. 2 1. def add(a, b): return a + b 2. result = add(3, 5) ``` Copied!
		Syntax: ``` 1. i 2. 2 1. try: # Code that might raise an exception except 2. ExceptionType: # Code to handle the exception ``` Copied!
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.	Example: ``` 1. i 2. 2 3. 3 4. 4 1. try: 2. num = int(input("Enter a number: ")) 3. except ValueError: 4. print("Invalid input. Please enter a valid number.") ``` Copied!
		Syntax: ``` 1. i 2. 2 3. 3 1. try: # Code that might raise an exception except 2. ExceptionType: # Code to handle the exception 3. else: # Code to execute if no exception occurs ``` Copied!
Try-Except with Else Block	Code in the `else` block is executed if no exception occurs in the try block.	Example: ``` 1. i 2. 2 3. 3 4. 4 5. 5 6. 6 1. try: 2. num = int(input("Enter a number: ")) 3. except ValueError: 4. print("Invalid input. Please enter a valid number") 5. else: 6. print("You entered:", num) ``` Copied!
		Syntax: ``` 1. i 2. 2 3. 3 1. try: # Code that might raise an exception except 2. ExceptionType: # Code to handle the exception 3. finally: # Code that always executes ``` Copied!
Try-Except with Finally Block	Code in the `finally` block always executes, regardless of whether an exception occurred.	Example: ``` 1. i 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 1. try: 2. file = open('data.txt', 'r') 3. data = file.read() 4. except FileNotFoundError: 5. print("File not found.") 6. finally: 7. file.close() ``` Copied!

While Loop	A 'while' loop repeatedly executes a block of code as long as a specified condition remains 'True'.

Syntax:

```
1. 1
```

```
1. while condition: # Code to repeat
```

Copied!

Example:

```
1. 1
2. 2
```

```
1. count = 0 while count < 5:
2.     print(count) count += 1
```

Copied!

Skills Network