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:
		
		```\nvariable1 == variable2\n```
		
		Example 1:
		
		```\n5 == 5\n```
		
		returns True
		
		Example 2:
		
		```\nage = 25\nage == 30\n```
		
		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:
		
		```\nfor variable in sequence: # Code to repeat\n```
		
		Example 1:
		
		```\nfor num in range(1, 10):\n print(num)\n```
		
		Example 2:
		
		```\nfruits = ["apple", "banana", "orange", "grape", "kiwi"]\nfor fruit in fruits:\n print(fruit)\n```

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 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 = 20
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: ``` size = 38 max_size = 40 size <= max_size ``` returns True

		Syntax:
| | | ```
variable1 < variable2
``` |
| Less Than(<) | Checks if the value of variable1 is less than variable2. | Example 1:<br><br>```
4 < 6
```<br><br>returns True<br><br>Example 2:<br><br>```
score = 60
passing_score = 65
score < passing_score
```<br><br>returns True |
| Loop Controls | `break` exits the loop prematurely. `continue` skips the rest of the current iteration and moves to the next iteration. | Syntax:<br><br>```
for: # Code to repeat
    if # boolean statement
        break
for: # Code to repeat
    if # boolean statement
        continue
```<br><br>Example 1:<br><br>```
for num in range(1, 6):
    if num == 3:
        break
    print(num)
```<br><br>Example 2:<br><br>```
for num in range(1, 6):
    if num == 3:
        continue
    print(num)
``` |

NOT	Returns `True` if variable is `False`, and vice versa.	Syntax:
		
		```\nnot variable\n```
		
		Example:
		
		```\nisLocked = False\nprint(not isLocked)\n```
		
		returns True if the variable is False (i.e., unlocked).

Not Equal(!=)	Checks if two values are not equal.	Syntax:
		
		```\nvariable1 != variable2\n```
		
		Example:
		
		```\na = 10\nb = 20\na != b\n```
		
		returns True
		
		Example 2:
		
		```\ncount=0\ncount != 0\n```
		
		returns False

Object Creation	Creates an instance of a class (object) using the class constructor.	Syntax:
		
		```\nobject_name = ClassName(arguments)\n```
		
		Example:
		
		```\nperson1 = Person("Alice", 25)\n```

OR	Returns `True` if either statement1 or statement2 (or both) are `True`. Otherwise, returns `False`.	Syntax:
		
		```\nstatement1 or statement2\n```
		
		Example:
		
		```\n"Farewell Party Invitation"\ngrade = 12\nif grade == 11 or grade == 12:\n print("Farewell Party Invitation")\nelse:\n print("Not eligible")\n```
		
		returns True

range()	Generates a sequence of numbers within a specified range.	Syntax:
		
		```\nrange(stop)\nrange(start, stop)\nrange(start, stop, step)\n```
		
		Example:
		
		```\nrange(5) #generates a sequence of integers from 0 to 4.\nrange(2, 10) #generates a sequence of integers from 2 to 9.\nrange(1, 11, 2) #generates odd integers from 1 to 9.\n```

Return Statement	`Return` is a keyword used to send a value back from a function to its caller.	Syntax:
		
		```\nreturn value\n```
		
		Example:
		
		```\ndef add(a, b): return a + b\nresult = add(3, 5)\n```

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:
		
		```\ntry: # Code that might raise an exception except\nExceptionType: # Code to handle the exception\n```
		
		Example:
		
		```\ntry:\n num = int(input("Enter a number: "))\nexcept ValueError:\n print("Invalid input. Please enter a valid number.")\n```

Try-Except with Else Block	Code in the `else` block is executed if no exception occurs in the try block.	Syntax:
		
		```\ntry: # Code that might raise an exception except\nExceptionType: # Code to handle the exception\nelse: # Code to execute if no exception occurs\n```
		
		Example:
		
		```\ntry:\n num = int(input("Enter a number: "))\nexcept ValueError:\n print("Invalid input. Please enter a valid number")\nelse:\n print("You entered:", num)\n```

Try-Except with Finally Block	Code in the `finally` block always executes, regardless of whether an exception occurred.	Syntax:
		
		```\ntry: # Code that might raise an exception except\nExceptionType: # Code to handle the exception\nfinally: # Code that always executes\n```
		
		Example:
		
		```\ntry:\n file = open("data.txt", "r")\n data = file.read()\nexcept FileNotFoundError:\n print("File not found.")\nfinally:\n file.close()\n```

While Loop	A `while` loop repeatedly executes a block of code as long as a specified condition remains `True`.	Syntax:
		
		```\nwhile condition: # Code to repeat\n```
		
		Example:
		
		```\ncount = 0\nwhile count < 5:\n print(count)\n count += 1\n```

Skills Network