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:
		
		```\nstatement1 and statement2\n```
		
		Example:
		
		```\nmarks = 90\nattendance_percentage = 87\nif marks >= 80 and attendance_percentage >= 85:\n print("qualify for honors")\nelse:\n print("Not qualified for honors")\n# Output = qualify for honors\n```

Class Definition	Defines a blueprint for creating objects and defining their attributes and behaviors.	Syntax:
		
		```\nclass ClassName: # Class attributes and methods\n```
		
		Example:
		
		```\nclass Person:\n def __init__(self, name, age):\n self.name = name\n self.age = age\n```

Define Function	A `function` is a reusable block of code that performs a specific task or set of tasks when called.	Syntax:
		
		```\ndef function_name(parameters): # Function body\n```
		
		Example:
		
		```\ndef greet(name): print("Hello,", name)\n```

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
```<br><br><br><br><br><br>returns False |
| If Statement | Executes code block `if` the condition is `True`. | Syntax:<br><br>```
if condition: #code block for if statement
```<br><br><br><br><br><br>Example:<br><br>```
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:<br><br>```
if condition1:
# Code if condition1 is True
elif condition2:
# Code if condition2 is True
else:
# Code if no condition is True
```<br><br><br><br>Example:<br><br>```
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:<br><br>```
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

Less Than(<)	Checks if the value of variable1 is less than variable2.	Syntax:
		
		`variable1 < variable2`
		
		Example 1:
		
		`4 < 6`
		
		returns True

		Example 2:
		
		``python
		score = 60
		passing_score = 65
		score < passing_score
		``
		
		returns True

Loop Controls	`break` exits the loop prematurely. `continue` skips the rest of the current iteration and moves to the next iteration.	Syntax:
		
		``python
		for: # Code to repeat
		    if # boolean statement
		        break
		for: # Code to repeat
		    if # boolean statement
		        continue
		``
		
		Example 1:
		
		``python
		for num in range(1, 6):
		    if num == 3:
		        break
		    print(num)
		``
		
		Example 2:
		
		``python
		for num in range(1, 6):
		    if num == 3:
		        continue
		    print(num)
		``

NOT	Returns `True` if variable is `False`, and vice versa.	Syntax:
		
		``python
		not variable
		``
		
		Example:
		
		``python
		isLocked = False
		print(not isLocked)
		``

		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:
		
		``
		"Farewell Party Invitation"
		grade = 12
		if grade == 11 or grade == 12:
		    print("Farewell Party Invitation")
		else:
		    print("Not eligible")
		``
		
		returns True

range()	Generates a sequence of numbers within a specified range.	Syntax: ``` range(stop) range(start, stop) range(start, stop, step) ``` Example: ``` 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. ```
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:<br><br>```
try: # Code that might raise an exception except
ExceptionType: # Code to handle the exception
else: # Code to execute if no exception occurs
```<br><br>Example:<br><br>```
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. | Syntax:<br><br>```
try: # Code that might raise an exception except
ExceptionType: # Code to handle the exception
finally: # Code that always executes
```<br><br>Example:<br><br>```
try:
    file = open("data.txt", "r")
    data = file.read()
except FileNotFoundError:
    print("File not found.")
finally:
    file.close()
``` |
| While Loop | A `while` loop repeatedly executes a block of code as long as a specified condition remains `True`. | Syntax:<br><br>```
while condition: # Code to repeat
``` |

Example:

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