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`
		Copied!
		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`
		Copied!

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

Define Function	A \`function\` is a reusable block of code that performs a specific task or	Syntax: `def function_name(parameters): # Function body`

set of tasks when called.

Copied!

Example:

```
def greet(name): print("Hello,", name)
```

Copied!

Equal(==)

Checks if two values are equal.

Syntax:

```
variable1 == variable2
```

Copied!

Example 1:

```
5 == 5
```

Copied!

returns True

Example 2:

```
age = 25 age == 30
```

Copied!

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
```

Copied!

Example 1:

```
for num in range(1, 10):
    print(num)
```

Copied!

Example 2:

```
fruits = ["apple", "banana", "orange", "grape", "kiwi"]
```

		`for fruit in fruits:`
		`    print(fruit)`
		Copied!

Function Call	A function call is the act of executing the code within the function using the provided arguments.	Syntax:
		`function_name(arguments)`
		Copied!
		Example:
		`greet("Alice")`
		Copied!

Greater Than or Equal To(>=)	Checks if the value of variable1 is greater than or equal to variable2.	Syntax:
		`variable1 >= variable2`
		Copied!
		Example 1:
		`5 >= 5 and 9 >= 5`
		Copied!
		returns True
		Example 2:
		`quantity = 105`
		`minimum = 100`
		`quantity >= minimum`
		Copied!
		returns True

Greater Than(>)	Checks if the value of variable1 is greater than variable2.	Syntax: `variable1 > variable2` Copied! Example 1: 9 > 6 returns True

		Example 2:
		`age = 20`
		`max_age = 25`
		`age > max_age`
		Copied!
		returns False

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

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`
		Copied!
		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.
```

Copied!

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
```

Copied!

Example:

```
if age >= 18:
    print("You're an adult.")
else:
    print("You're not an adult yet.")
```

Copied!

Less Than or Equal To(<=)

Checks if the value of variable1 is less than or equal to variable2.

Syntax:

```
variable1 <= variable2
```

Copied!

Example 1:

```
5 <= 5 and 3 <= 5
```

Copied!

returns True

Example 2:

```
size = 38
max_size = 40
```

```
size <= max_size
```

Copied!

returns True

Less Than(<)	Checks if the value of variable1 is less than variable2.	Syntax: `variable1 < variable2` Copied! Example 1: `4 < 6` Copied! returns True Example 2: `score = 60` `passing_score = 65` `score < passing_score` Copied! returns True
Loop Controls	\`break\` exits the loop prematurely. \`continue\` skips the rest of the current iteration and moves to the next iteration.	Syntax: (see below)

Less Than(<) — Syntax:

```
variable1 < variable2
```

Copied!

Example 1:

```
4 < 6
```

Copied!

returns True

Example 2:

```
score = 60
passing_score = 65
score < passing_score
```

Copied!

returns True

Loop Controls — Syntax:

```
for: # Code to repeat
    if # boolean statement
        break

for: # Code to repeat
    if # boolean statement
        continue
```

Copied!

Example 1:

```
for num in range(1, 6):
    if num == 3:
        break
    print(num)
```

Copied!

Example 2:

```
for num in range(1, 6):
    if num == 3:
        continue
    print(num)
```

Copied!

NOT	Returns `True` if variable is `False`, and vice versa.	Syntax:
		`!variable`
		Copied!
		Example:
		`!isLocked`
		Copied!
		returns True if the variable is False (i.e., unlocked).

Not Equal(!=)	Checks if two values are not equal.	Syntax:
		`variable1 != variable2`
		Copied!
		Example:
		`a = 10`
		`b = 20`
		`a != b`
		Copied!
		returns True
		Example 2:

		`count=0`
		`count != 0`
		Copied!
		returns False

Object Creation	Creates an instance of a class (object) using the class constructor.	Syntax:
		`object_name = ClassName(arguments)`
		Copied!
		Example:
		`person1 = Person("Alice", 25)`
		Copied!

OR	Returns `True` if either statement1 or statement2 (or both) are `True`. Otherwise, returns `False`.	Syntax:
		`statement1 || statement2`
		Copied!
		Example:
		`"Farewell Party Invitation"`
		`Grade = 12 grade == 11 or grade == 12`
		Copied!
		returns True

range()	Generates a sequence of numbers within a specified range.	Syntax:
		`range(stop)`
		`range(start, stop)`
		`range(start, stop, step)`
		Copied!
		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.
```

Copied!

Return Statement

`Return` is a keyword used to send a value back from a function to its caller.

Syntax:

```
return value
```

Copied!

Example:

```
def add(a, b): return a + b
result = add(3, 5)
```

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.

Syntax:

```
try: # Code that might raise an exception except
ExceptionType: # Code to handle the exception
```

Copied!

Example:

```
try:
    num = int(input("Enter a number: "))
except ValueError:
    print("Invalid input. Please enter a valid number.")
```

Copied!

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
```

Copied!

Example:

```
try:
    num = int(input("Enter a number: "))
except ValueError:
    print("Invalid input. Please enter a valid number")
else:
    print("You entered:", num)
```

Copied!

Try-Except with Finally Block

Code in the `finally` block always executes, regardless of whether an exception occurred.

Syntax:

```
try: # Code that might raise an exception except
ExceptionType: # Code to handle the exception
finally: # Code that always executes
```

Copied!

Example:

```
try:
    file = open("data.txt", "r")
    data = file.read()
except FileNotFoundError:
    print("File not found.")
finally:
    file.close()
```

Copied!

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
```

Copied!

Example:

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

Copied!