# 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:<br>`statement1 and statement2`<br>Example:<br>`marks = 90`<br>`attendance_percentage = 87`<br>`if marks >= 80 and attendance_percentage >= 85:`<br>`print("qualify for honors")`<br>`else:`<br>`print("Not qualified for honors")`<br>`# Output = qualify for honors` |
| Class Definition | Defines a blueprint for creating objects and defining their attributes and behaviors. | Syntax:<br>`class ClassName: # Class attributes and methods`<br>Example:<br>`class Person:`<br>`def __init__(self, name, age):`<br>`self.name = name`<br>`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:<br>`def function_name(parameters): # Function body`<br>Example:<br>`def greet(name): print("Hello,", name)` |
| Equal(==) | Checks if two values are equal. | Syntax:<br>`variable1 == variable2`<br>Example 1:<br>`5 == 5`<br>returns True<br>Example 2:<br>`age = 25 age == 30`<br>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:<br>`for variable in sequence: # Code to repeat`<br>Example 1:<br>`for num in range(1, 10):`<br>`print(num)`<br>Example 2:<br>`fruits = ["apple", "banana", "orange", "grape", "kiwi"]`<br>`for fruit in fruits:`<br>`print(fruit)` |
| Function Call | A function call is the act of executing the code within the function using the provided arguments. | Syntax:<br>`function_name(arguments)`<br>Example:<br>`greet("Alice")` |
| Greater Than or Equal To(>=) | Checks if the value of variable1 is greater than or equal to variable2. | Syntax:<br>`variable1 >= variable2`<br>Example 1:<br>`5 >= 5 and 9 >= 5`<br>returns True<br>Example 2:<br>`quantity = 105`<br>`minimum = 100`<br>`quantity >= minimum` |

| | | |
|---|---|---|
| | | returns True |
| Greater Than(>) | Checks if the value of variable1 is greater than variable2. | Syntax:<br>`variable1 > variable2`<br>Example 1: 9 > 6<br>returns True<br>Example 2:<br>`age = 20`<br>`max_age = 25`<br>`age > max_age`<br>returns False |
| If Statement | Executes code block `if` the condition is `True`. | Syntax:<br>`if condition: #code block for if statement`<br>Example:<br>`if temperature > 30:`<br>`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>`if condition1:`<br>`# Code if condition1 is True`<br>`elif condition2:`<br>`# Code if condition2 is True`<br>`else:`<br>`# Code if no condition is True`<br>Example:<br>`score = 85  # Example score`<br>`if score >= 90:`<br>`print("You got an A!")`<br>`elif score >= 80:`<br>`print("You got a B.")`<br>`else:`<br>`print("You need to work harder.")`<br>`# Output = You got a B.` |
| If-Else Statement | Executes the first code block if the condition is `True`, otherwise the second block. | Syntax:<br>`if condition: # Code, if condition is True`<br>`else: # Code, if condition is False`<br>Example:<br>`if age >= 18:`<br>`print("You're an adult.")`<br>`else:`<br>`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:<br>`variable1 <= variable2`<br>Example 1:<br>`5 <= 5 and 3 <= 5`<br>returns True<br>Example 2:<br>`size = 38`<br>`max_size = 40`<br>`size <= max_size`<br>returns True |
| Less Than(<) | Checks if the value of variable1 is less than variable2. | Syntax:<br>`variable1 < variable2`<br>Example 1:<br>`4 < 6`<br>returns True<br>Example 2:<br>`score = 60`<br>`passing_score = 65`<br>`score < passing_score`<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>`for: # Code to repeat`<br>`  if # boolean statement`<br>`  break`<br>`for: # Code to repeat`<br>`  if # boolean statement`<br>`  continue`<br>Example 1:<br>`for num in range(1, 6):`<br>`  if num == 3:`<br>`  break`<br>`  print(num)`<br>Example 2:<br>`for num in range(1, 6):`<br>`  if num == 3:`<br>`  continue`<br>`  print(num)` |
| NOT | Returns `True` if variable is `False`, and vice versa. | Syntax:<br>`!variable`<br>Example:<br>`!isLocked`<br>returns True if the variable is False (i.e., unlocked). |
| Not Equal(!=) | Checks if two values are not equal. | Syntax:<br>`variable1 != variable2`<br>Example:<br>`a = 10`<br>`b = 20`<br>`a != b`<br>returns True<br>Example 2:<br>`count=0`<br>`count != 0`<br>returns False |
| Object Creation | Creates an instance of a class (object) using the class constructor. | Syntax:<br>`object_name = ClassName(arguments)`<br>Example:<br>`person1 = Person("Alice", 25)` |
| OR | Returns `True` if either statement1 or statement2 (or both) are `True`. Otherwise, returns `False`. | Syntax:<br>`statement1 \|\| statement2`<br>Example:<br>`"Farewell Party Invitation"`<br>`Grade = 12 grade == 11 or grade == 12`<br>returns True |
| range() | Generates a sequence of numbers within a specified range. | Syntax:<br>`range(stop)`<br>`range(start, stop)`<br>`range(start, stop, step)`<br>Example:<br>`range(5) #generates a sequence of integers from 0 to 4.`<br>`range(2, 10) #generates a sequence of integers from 2 to 9.`<br>`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:<br>`return value`<br>Example:<br>`def add(a, b): return a + b`<br>`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 | Syntax:<br>`try: # Code that might raise an exception except` |

| | | |
|---|---|---|
| | executed. | `ExceptionType: # Code to handle the exception`<br><br>Example:<br><br>`try:`<br>`  num = int(input("Enter a number: "))`<br>`except ValueError:`<br>`  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:`<br>`  num = int(input("Enter a number: "))`<br>`except ValueError:`<br>`  print("Invalid input. Please enter a valid number")`<br>`else:`<br>`  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:`<br>`  file = open("data.txt", "r")`<br>`  data = file.read()`<br>`except FileNotFoundError:`<br>`  print("File not found.")`<br>`finally:`<br>`  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`<br><br>Example:<br><br>`count = 0 while count < 5:`<br>`  print(count) count += 1` |

Skills Network