Keywords & Variables

Keywords

Keywords are reserved words in Python that have predefined meanings and cannot be used as variable names or identifiers. These are used to define the structure and logic of the program.

Below are some important Python keywords:

1. Logical Operators:

- **and:** The and operator only gives True when both sides (conditions) are True. If either one is False, the result will be False.
- **or:** The or operator gives True if at least one of the conditions is True. It returns False if both the conditions are False.
- **not:** It is a logical operator that reverse the truth value of a Boolean expression. It's commonly used when you want to check if something is **not** true.
  e.g –

```
x = 5
y = 10
if not x == y:
    print("x is not equal to y")                    # Output - x is not equal to y
```

2. Conditional Statements:

- **if:** It executes a block of code if the given condition is true.
- **else:** If the condition in the if statement is not true, the else block will be executed.
- **elif:** The elif statement (stands for else if, but in python we use elif) is used when you have multiple conditions to check, and you want to test a new condition if the previous if and else conditions are False**.**
  e**.**g –

```
age = 19

if age >= 21:
    print("Eligible to vote and above 21.")
elif age >= 18:
    print("Eligible to vote but under 21.")
else:
    print("Not eligible to vote.")          # Output - Eligible to vote but under 21.
```

3. Loops:

- **while:** Used to create a loop that executes as long as a specified condition is True.
- **for:** Used to repeat an action a certain number of times or for each item in a collection (like a list or string).
- **In:** Used with for to check if a value is present in a sequence (like a list, tuple, or string).

4. Exception Handling: Exception handling allow us to handle errors gracefully and avoid any program crashing.

- **try:** This block of code is where you place the code that might cause an error.
- **except:** This block handles the error if something goes wrong in the try block.
- **finally:** This block always runs, regardless of whether there was an error or not.
  e.g –

```
try:
    # Code that might cause an error
    result = 10 / 0
    print("This won't print because of the error.")
except ZeroDivisionError:                    # This block handles the error
    print("You can't divide by zero!")
finally:
    print("This will run no matter what!")
```

Output - You can't divide by zero!
This will run no matter what!

5. Function Definition:

- **def:** Used to define a function in Python.
  e.g of def –

```
a=20
b=10

def add():
    print(a+b)

def sub():
    print(a-b)

add()
sub()
```

Output –
30
10

- **return:** Used within a function to specify the value to return.
  e.g –

```
a = 20              # Global Variable define (it will be used throughout the program)
b = 10

def add():
   return a + b  # Return the sum of a and b

def sub():
   return a - b  # Return the difference of a and b

# Call the functions and store the results
result_add = add()  # Store the result of addition
result_sub = sub()  # Store the result of subtraction

# Print the results
print("Addition Result:", result_add)
print("Subtraction Result:", result_sub)
```

6. Object-Oriented Programming (OOP):

- **class:** Used to define a class, which is a blueprint for creating objects in object-oriented programming.

7. Importing Modules:

- **import:** Used to import external modules or libraries to access their functions, classes, or variables.
- **from:** Used with import to specify which specific components from a module should be imported.
- **as:** Used with import to create an alias for a module, making it easier to reference.

8. Boolean Values and Constants:

- **True:** Represents a boolean value for True.
- **False:** Represents a boolean value for False.
- **None:** Represents a special null value or the absence of a value.

9. Identity and Comparison:

- **is:** Used for identity comparison, checking if two variables refer to the same object in memory.

10. Lambda Functions:

- **lambda:** Used to create small, anonymous functions (lambda functions).

11. Context Management:

- **with:** Used for context management to ensure certain operations are performed before and after a block of code.

12. Variable Scope Management:

- **global:** Used to declare a global variable within a function's scope.
- **nonlocal:** Used to declare a variable as nonlocal, which allows modifying a variable in an enclosing (but non-global) scope.

Variables

In Python, a **variable** is essentially a name that refers to a value. When you assign a value to a variable, Python creates a reference to that value, which can be used later in your code. Below are few e.g of defining variable -

- server_name = "my_server"
- port = 80
- is_https_enabled = True
- max_connections = 1000

Variable Naming Rules –

- They must start with a letter (a-z, A-Z) or an underscore (_).
- The rest of the variable name can contain letters, numbers (0-9), and underscores.
- Variable names are **case-sensitive** (myvar is different from MyVar).
- Reserved words (like class, if, while, for, etc.) cannot be used as variable names
- **Snake Casing** – Snake casing is a way of defining variable name. It is used when variable name has multiple parts, use "_" between each part , e.g –
  Ec2_instance_name = "Production Server"
- **Camel Casing** – Camel casing is also a naming convention of variable in which –
  - The first word starts with a lowercase letter, while each subsequent word starts with an uppercase letter. There are no underscores between the words.
    e,g – ec2InstanceName = "Production Server"

Scope of Variables

Global **variables** are defined outside of functions and are accessible from anywhere in the program.
Local **variables** are defined within functions and are only accessible inside that function

E.g of Global Variable declaration – already seen in Function Definition in return above
E.g of Local Variable –

```
def add():
    a=10                    # variable defined within a function.
    b=10
    print(a+b)

add()
```

Example: Using Variables to Store and Manipulate Configuration Data in a DevOps Context

```
# Define configuration variables for a web server
server_name = "my_server"
port = 80
is_https_enabled = True
max_connections = 1000

# Print the configuration
print(f"Server Name: {server_name}")
print(f"Port: {port}")
print(f"HTTPS Enabled: {is_https_enabled}")
print(f"Max Connections: {max_connections}")

# Update configuration values
port = 443
is_https_enabled = False

# Print the updated configuration
print(f"Updated Port: {port}")
print(f"Updated HTTPS Enabled: {is_https_enabled}")
```

Output -

```
Server Name: my_server
Port: 80
HTTPS Enabled: True
Max Connections: 1000
Updated Port: 443
Updated HTTPS Enabled: False
```