

Day 1 - Keywords

Premanand S

Assistant Professor
School of Electronics Engineering
Vellore Institute of Technology
Chennai Campus

premanand.s@vit.ac.in

October 19, 2024

What is a Keyword?

Definition: A **keyword** in Python is a reserved word that has a predefined meaning and purpose within the Python language.

- Keywords are part of the Python syntax.
- They cannot be used as identifiers (variable names, function names, etc.).

Examples: if, else, while, def, etc.

Importance of Using Keywords

1. Syntax and Structure:

- Keywords define the core structure and rules for Python code.

2. Readability:

- Keywords make the code more readable and easier to understand for others.

3. Control Flow:

- Keywords like `if`, `else`, `for`, `while` control the program's logic.

4. Memory Management:

- Keywords like `del` help manage memory by deleting unused objects.

Importance of Using Keywords (continued)

5. Exception Handling:

- Keywords like `try`, `except`, and `finally` make your program more robust by handling exceptions.

6. Object-Oriented Programming (OOP):

- Keywords like `class`, `self`, and `super` help create and manipulate objects and classes.

Characteristics of Python Keywords

1. Reserved Words:

- Keywords cannot be used as identifiers (e.g., variable names).

2. Predefined Behavior:

- Each keyword has a specific purpose that defines the behavior of the program.

3. Case-Sensitive:

- Python keywords are case-sensitive, for example, `True` is a keyword, but `true` is not.

4. Fixed in Number:

- Python has a fixed number of keywords, which can change with different Python versions.

5. Essential to Python Syntax:

- Keywords form the core of Python's syntax and structure for loops, functions, conditions, and more.

Access Keyword: Ananconda Command Prompt

```
Anaconda Prompt - python
(base) C:\Users\S.A.N>python
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> help()

Welcome to Python 3.11's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the internet at https://docs.python.org/3.11/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False          class           from            or
None           continue       global          pass
True           def            if              raise
and            del            import          return
as             elif           in              try
assert         else           is              while
async          except         lambda          with
await          finally        nonlocal        yield
break          for            not

help> |
```

Access Keyword: import keyword

- The keyword module provides the list of reserved Python keywords.

Example (Python Code)

```
# importing modules for getting keywords
import keyword

# Get the list of all Python keywords
all_keywords = keyword.kwlist

# Display the keywords
print("Python Keywords:")
for kw in all_keywords:
    print(kw)
```


Python Keywords List

The following are all reserved keywords in Python:

- and
- as
- assert
- async
- await
- break
- class
- continue
- def
- del
- elif
- else
- except
- False
- finally
- for
- from
- global
- if
- import
- in
- is
- lambda
- None
- nonlocal
- not
- or
- pass
- raise
- return
- True
- try
- while
- with
- yield

Keyword: False (Boolean Value)

- Represents the boolean value False in Python.
- Used to indicate that a condition or expression evaluates to False.

Example (Python Code)

```
a = False
if a:
    print("The condition is True")
else:
    print("The condition is False")
```

Output:

The condition is False

Keyword: None (Null Value)

- None is a special constant in Python representing the absence of a value or a null value.
- It is not the same as False, 0, or an empty string.
- Used to indicate that a variable has no value or to reset a variable.

Example (Python Code)

```
a = None
if a is None:
    print("Variable has no value")
else:
    print("Variable has a value")
```

Output:

Variable has no value

Keyword: True (Boolean Value)

- True represents the boolean value of true in Python.
- It is often used in conditional statements to control the flow of the program.

Example (Python Code)

```
a = True
if a:
    print("The condition is True")
else:
    print("The condition is False")
```

Output:

The condition is True

Keyword: and (Logical Operator)

- Combines two conditions and returns True if both conditions are true.
- Often used in control flow statements like `if` and `while`.

Example (Python Code)

```
a = True
b = False
if a and b:
    print("Both are True")
else:
    print("At least one is False")
```

Output:

At least one is False

Keyword: as (Alias Keyword)

- The as keyword is used to create an alias for a module or a function.
- It allows you to reference modules or functions with a shorter or more convenient name.
- Commonly used in import statements and with blocks.

Example (Python Code)

```
import math as m
print(m.sqrt(16))    # Using the alias 'm'

with open("example.txt", "w") as file:
    file.write("Hello, World!")
```

Output:

4.0

Keyword: assert (Debugging Aid)

- The assert keyword is used for debugging.
- It tests if a condition is True; if not, it raises an AssertionError.
- Typically used to catch bugs or check assumptions in code.

Example (Python Code)

```
x = 10
assert x > 5, "x should be greater than 5"
assert x < 5, "x should be less than 5"
print("All assertions passed")
```

Output:

Traceback (most recent call last):

File "example.py", line 3, in <module>

assert x < 5, "x should be less than 5"

AssertionError: x should be less than 5

Keyword: `async` (Asynchronous Programming)

- The `async` keyword is used to define an asynchronous function
- It allows you to write functions that can pause and resume their execution to handle other tasks, improving efficiency for I/O-bound operations.
- Used together with `await` to perform non-blocking operations.

Example (Python Code)

```
import asyncio
async def greet():
    print("Hello!")
    await asyncio.sleep(1)
    print("World!")
asyncio.run(greet())
```

Output:

Hello!

World!

Keyword: await (Asynchronous Programming)

- The await keyword is used inside an async function to pause its execution until the awaited task is completed.
- Typically used to wait for asynchronous operations such as I/O tasks, network requests, or time delays.

Example (Python Code)

```
import asyncio
async def count_down():
    print("Starting countdown...")
    await asyncio.sleep(2) # Pausing for 2 seconds
    print("Countdown finished!")
asyncio.run(count_down())
```

Output:

```
Starting countdown...
(2-second delay)
Countdown finished!
```

Keyword: break (Loop Control)

- The break keyword is used to exit a loop prematurely.
- It immediately terminates the enclosing loop (for or while) when a specified condition is met.
- Commonly used in loops to stop execution once a certain condition is true.

Example (Python Code)

```
for i in range(5):  
    if i == 3:  
        break  
    print(i)
```

Output:

0
1
2

Keyword: class (Object-Oriented Programming)

- The `class` keyword is used to define a class in Python, which is a blueprint for creating objects.
- Classes encapsulate data (attributes) and methods (functions) that operate on the data

Example (Python Code)

```
class Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed
    def bark(self):
        return f"{self.name} says Woof!"
my_dog = Dog("Buddy", "Golden Retriever")
print(my_dog.bark())
```

Output:

Buddy says Woof!

Keyword: continue (Loop Control)

- The `continue` keyword is used to skip the current iteration of a loop and move to the next iteration.
- It is often used to avoid executing certain parts of the loop based on a condition.

Example (Python Code)

```
for i in range(5):  
    if i == 3:  
        continue  
    print(i)
```

Output:

0
1
2
4

Keyword: def (Function Definition)

- The def keyword is used to define a function in Python.
- It allows you to encapsulate a block of code that can be executed when the function is called.
- Functions can take arguments, return values, and improve code reusability and organization.

Example (Python Code)

```
def greet(name):  
    return f"Hello, {name}!"  
  
# Calling the function  
message = greet("Prem")  
print(message)
```

Output:

Hello, Prem!

Keyword: del (Delete Statement)

- The del keyword is used to delete objects in Python. It can remove variables, list elements, or even entire dictionaries and their keys.
- Once an object is deleted, it can no longer be accessed.

Example (Python Code)

```
my_list = [1, 2, 3, 4, 5]
print("Original list:", my_list)
del my_list[2] # Removes the element at index 2
print("List after deletion:", my_list)
del my_list # Deleting the entire list
print("List deleted.")
```

Output:

```
Original list: [1, 2, 3, 4, 5]
List after deletion: [1, 2, 4, 5]
List deleted.
```

Keyword: elif (Conditional Statements)

- The `elif` keyword is short for "else if." It allows you to check multiple expressions for truthiness, following an initial `if` statement.
- You can have multiple `elif` blocks to handle different conditions

Example (Python Code)

```
x = 20
if x < 10:
    print("x is less than 10")
elif x < 20:
    print("x is less than 20")
elif x == 20:
    print("x is equal to 20")
else:
    print("x is greater than 20")
```

Output:

x is equal to 20

Keyword: else (Conditional Statements)

- The else keyword is used in conditional statements to define a block of code that executes when the conditions in the preceding if and elif statements are false.
- It acts as a default case when none of the specified conditions are met.

Example (Python Code)

```
x = 15
if x < 10:
    print("x is less than 10")
elif x < 20:
    print("x is less than 20")
else:
    print("x is greater than or equal to 20")
```

Output:

x is less than 20

Keyword: `except` (Exception Handling)

- The `except` keyword is used in try-except blocks to handle exceptions (errors) that may occur during the execution of a program.
- It allows you to define how your program should respond to specific errors, preventing the program from crashing.
- You can specify a particular exception type or use a generic `except` to catch all exceptions.

Keyword: except (Exception Handling)

Example (Python Code)

```
try:
    number = int(input("Enter a number: "))
    result = 100 / number
    print("Result:", result)
except ValueError:
    print("Please enter a valid integer.")
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
except:
    print("An unexpected error occurred.")
```

Output (if user inputs 0):

```
Enter a number: 0
Error: Division by zero is not allowed.
```

Keyword: `finally` (Exception Handling)

- The `finally` keyword is used in try-except blocks to define a block of code that will always execute, regardless of whether an exception was raised or handled.
- It is typically used for cleanup actions, such as closing files or releasing resources, ensuring that necessary steps are taken even if an error occurs.

Keyword: finally (Exception Handling)

Example (Python Code)

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

Output (if the file does not exist):

```
File not found.
Executing finally block.
```

Keyword: for (Looping)

- The for keyword is used to create a loop that iterates over a sequence (like a list, tuple, or string) or any iterable object.
- It allows you to execute a block of code repeatedly for each item in the sequence.

Example (Python Code)

```
fruits = ["apple", "banana", "cherry"]  
  
for fruit in fruits:  
    print(fruit)
```

Output:

```
apple  
banana  
cherry
```

Keyword: from (Import Statement)

- The from keyword is used in import statements to import specific functions, classes, or variables from a module.
- It allows for a more efficient way to access particular parts of a module without importing the entire module.

Example (Python Code)

```
from math import sqrt, pi

result = sqrt(16)

print("Square root of 16:", result)
print("Value of pi:", pi)
```

Output:

```
Square root of 16: 4.0
Value of pi: 3.141592653589793
```

Keyword: global (Variable Scope)

- The `global` keyword is used to declare a variable as global, allowing it to be accessed and modified inside a function.
- Without the `global` declaration, a variable assigned within a function is treated as a local variable, and any attempt to modify it will result in an error if it hasn't been declared.

Example (Python Code)

```
x = 10 # Global variable
def modify_global():
    global x # Declare x as global
    x += 5 # Modify the global variable
modify_global()
print("Value of x after modification:", x)
```

Output:

Value of x after modification: 15

Keyword: if (Conditional Statement)

- The `if` keyword is used to create a conditional statement that executes a block of code if a specified condition evaluates to `True`.
- It is fundamental for controlling the flow of the program based on dynamic conditions.

Example (Python Code)

```
age = 18

if age >= 18:
    print("You are an adult.")
```

Output:

You are an adult.

Keyword: import (Module Importing)

- The `import` keyword is used to include external modules in your Python program, giving access to their functions, classes, and variables.
- It allows for code reuse and modular programming, enabling you to leverage pre-built libraries and functionality.

Example (Python Code)

```
import math

# Calculate the square root of 25
result = math.sqrt(25)

# Print the result
print("Square root of 25:", result)
```

Output:

Square root of 25: 5.0

Keyword: in (Membership Operator)

- The `in` keyword is used to check for membership, determining whether a value exists within a sequence (like a list, tuple, or string).
- It returns `True` if the value is found in the sequence; otherwise, it returns `False`.

Example (Python Code)

```
fruits = ["apple", "banana", "cherry"]

# Check if "banana" is in the list
if "banana" in fruits:
    print("Banana is in the list.")
else:
    print("Banana is not in the list.")
```

Output:

Banana is in the list.

Keyword: is (Identity Operator)

- The `is` keyword is used to compare the identities of two objects, checking whether they refer to the same object in memory.
- It returns `True` if both operands refer to the same object; otherwise, it returns `False`.

Example (Python Code)

```
a = [1, 2, 3]
b = a # b references the same list as a
c = [1, 2, 3] # c is a different list with the same content

# Check identity
print("a is b:", a is b) # True
print("a is c:", a is c) # False
```

Output:

```
a is b: True
a is c: False
```

Keyword: lambda (Anonymous Function)

- The `lambda` keyword is used to create anonymous functions, which are small, unnamed functions defined in a single line.
- It can take any number of arguments but can only have one expression. The result of the expression is returned automatically.

Example (Python Code)

```
# Define a lambda function to calculate the square of a num
square = lambda x: x ** 2

# Use the lambda function
result = square(5)

# Print the result
print("Square of 5:", result)
```

Output:

Square of 5: 25

Keyword: `nonlocal` (Variable Scope)

- The `nonlocal` keyword is used to declare a variable in a nested function that refers to a variable in the nearest enclosing scope that is not global.
- It allows you to modify a variable from an outer (but not global) scope within a nested function.

Keyword: nonlocal (Variable Scope)

Example (Python Code)

```
def outer_function():  
    x = "local"  # x is a local variable in outer_function  
    def inner_function():  
        nonlocal x  # Declare x as nonlocal  
        x = "nonlocal"  # Modify the nonlocal variable  
    inner_function()  
    return x  
  
# Call the outer function  
result = outer_function()  
print("Value of x after inner_function:", result)
```

Output:

Value of x after inner_function: nonlocal

Keyword: not (Logical Operator)

- The not keyword is a logical operator used to negate a boolean value or expression.
- It returns True if the operand is False, and False if the operand is True.

Example (Python Code)

```
a = True
b = False

# Using the not operator
print("not a:", not a) # This will return False
print("not b:", not b) # This will return True
```

Output:

```
not a: False
not b: True
```

Keyword: or (Logical Operator)

- The or keyword is a logical operator that combines two conditions and returns True if at least one of the conditions is true.
- It is commonly used in control flow statements like if and while.

Example (Python Code)

```
a = True
b = False

# Using the or operator
if a or b:
    print("At least one is True")
else:
    print("Both are False")
```

Output:

At least one is True

Keyword: pass (Null Statement)

- The pass keyword is a null statement in Python. It serves as a placeholder in control flow structures where a statement is syntactically required but no action is desired.
- It is often used in function definitions, loops, or conditionals where you want to implement a feature later without raising an error.

Example (Python Code)

```
def my_function():  
    pass # Function does nothing for now  
  
# Use the function  
my_function()  
print("Function executed without errors.")
```

Output:

Function executed without errors.

Keyword: raise (Exception Handling)

- The `raise` keyword is used to trigger an exception in Python. It can be used to indicate that an error has occurred or to re-raise a caught exception.
- It allows developers to create custom error messages and manage exceptions effectively.

Keyword: raise (Exception Handling)

Example (Python Code)

```
def check_age(age):  
    if age < 0:  
        raise ValueError("Age cannot be negative")  
    return age  
  
try:  
    check_age(-1)  
except ValueError as e:  
    print("Caught an exception:", e)
```

Output:

Caught an exception: Age cannot be negative

Keyword: `return` (Function Return)

- The `return` keyword is used in a function to send back a value to the caller and exit the function.
- It can return multiple values as a tuple and is crucial for getting results from function calls.

Keyword: return (Function Return)

Example (Python Code)

```
def add_and_multiply(a, b):  
    sum_result = a + b  
    product_result = a * b  
    return sum_result, product_result  
# Call the function and unpack the returned values  
sum_value, product_value = add_and_multiply(3, 5)  
print("Sum:", sum_value)  
print("Product:", product_value)
```

Output:

Sum: 8

Product: 15

Keyword: try (Exception Handling)

- The try keyword is used to define a block of code to test for errors or exceptions during execution.
- It is often used in conjunction with except to handle potential exceptions gracefully without crashing the program.

Example (Python Code)

```
def divide(x, y):  
    try:  
        result = x / y # Attempt to divide x by y  
    except ZeroDivisionError:  
        return "Error: Cannot divide by zero!" # Handle divis  
    return result  
output = divide(10, 0)  
print(output)
```

Output:

Error: Cannot divide by zero!

Keyword: while (Looping Statement)

- The `while` keyword is used to create a loop that repeatedly executes a block of code as long as a specified condition is true.
- It is useful for situations where the number of iterations is not known beforehand.

Keyword: while (Looping Statement)

Example (Python Code)

```
count = 0
while count < 5:
    print("Count is:", count)
    count += 1 # Increment count by 1
print("Finished counting.")
```

Output:

```
Count is: 0
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Finished counting.
```


Keyword: with (Context Management)

- The with keyword is used to wrap the execution of a block of code within methods defined by a context manager.
- It simplifies exception handling by encapsulating common preparation and cleanup tasks.

Example (Python Code)

```
# Using with statement to handle file operations
with open("example.txt", "w") as file:
    file.write("Hello, World!") # Writing to the file

# The file is automatically closed after the with block
print("File written successfully.")
```

Output:

```
File written successfully.
```

Keyword: `yield` (Generator Function)

- The `yield` keyword is used to create a generator function, which allows you to return an iterator that produces values one at a time, instead of returning them all at once.
- It enables the function to maintain its state between calls, making it memory-efficient for large datasets.

Keyword: yield (Generator Function)

Example (Python Code)

```
def count_up_to(n):  
    count = 1  
    while count <= n:  
        yield count # Yield the current count  
        count += 1 # Increment count  
counter = count_up_to(5)  
for number in counter:  
    print("Count:", number)
```

Output:

```
Count: 1  
Count: 2  
Count: 3  
Count: 4  
Count: 5
```

Exploring Python Keywords

- Understanding Python keywords may seem heavy at first, but they will become familiar with time and practice.
- To get a clearer understanding of how the code works, consider the following:
 - Copy the code examples provided.
 - Use the Thonny IDE to visualize code execution step by step.
 - Thonny is user-friendly and ideal for beginners, making it easier to grasp concepts.

mail me: er.anandprem@gmail.com / premanand.s@vit.ac.in
ring me: +91 73586 79961
follow me: LinkedIn
author at Analytics Vidhya: premanand17
instagram: premsanand

**Learning gives Creativity,
Creativity leads to Thinking,
Thinking provides Knowledge,
and
Knowledge makes you Great
- Dr APJ Abdul Kalam**