Python Introduction

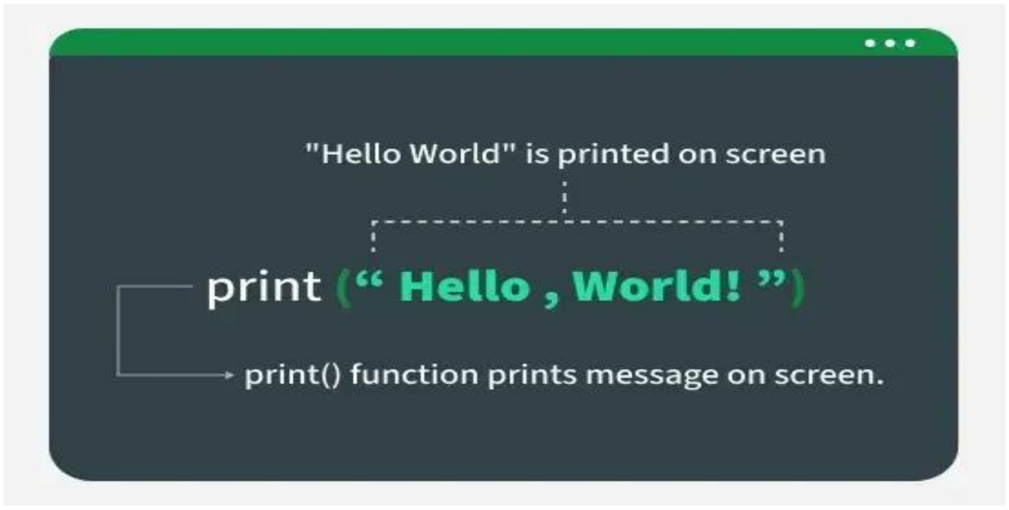
Python was created 1991 with focus on code readability and express concepts in fewer lines of code.

- Simple and readable syntax makes it beginner-friendly.
- Runs seamlessly on Windows, macOS and Linux.
- Includes libraries for tasks like web development, data analysis and machine learning.
- Variable types are determined automatically at runtime, simplifying code writing.
- Supports multiple programming paradigms, including object-oriented, functional and procedural programming.
- Free to use, distribute and modify.

Understanding Hello World Program in Python

Hello, World! in python is the first python program which we learn when we start learning any program. It's a simple program that displays the message "Hello, World!" on the screen.

Hello World Program



```
print (" Hello , World! ")
```

"Hello World" is printed on screen
print() function prints message on screen.

Here's the "Hello World" program:

```
# This is a comment. It will not be executed.

print("Hello, World!")
```

Output

```
Hello, World!
```

How does this work:

- print() is a built-in Python function that tells the computer to show something on the screen.
- The message "Hello, World!" is a string, which means it's just text. In Python, strings are always written inside quotes (either single ' or double ").
- Anything after # in a line is a comment. Python ignores comments when running the code, but they help people understand what the code is doing.
- Comments are helpful for explaining code, making notes or skipping lines while testing.

We can also write multi-line comments using triple quotes:

```
"""

This is a multi-line comment.

It can be used to describe larger sections of code.

"""
```

To understand comments in detail, refer to article: ***Comments.***

Indentation in Python

In Python, Indentation is used to define blocks of code. It tells the Python interpreter that a group of statements belongs to a specific block. All statements with the same level of indentation are considered part of the same block. Indentation is achieved using whitespace (spaces or tabs) at the beginning of each line. The most common convention is to use 4 spaces or a tab, per level of indentation.

Example:

```
print("I have no indentation")


  print("I have tab indentaion")
```

Output:

```
Hangup (SIGHUP)
File "/home/guest/sandbox/Solution.py", line 3
print("I have tab indentaion")
IndentationError: unexpected indent
```

Explanation:

- first **print** statement has no indentation, so it is correctly executed.
- second **print** statement has **tab indentation**, but it doesn't belong to a new block of code. Python expects the indentation level to be consistent within the same block. This inconsistency causes an **IndentationError**.

To understand Indentation in detail, refer to article: ***Indentation***

Famous Application Built using Python

- **YouTube:** World's largest video-sharing platform uses Python for features like video streaming and backend services.
- **Instagram:** This popular social media app relies on Python's simplicity for scaling and handling millions of users.
- **Spotify:** Python is used for backend services and machine learning to personalize music recommendations**.**
- **Dropbox:** The file hosting service uses Python for both its desktop client and server-side operations.
- **Netflix:** Python powers key components of Netflix's recommendation engine and content delivery systems (CDN).
- **Google:** Python is one of the key languages used in Google for web crawling, testing and data analysis.
- **Uber:** Python helps Uber handle dynamic pricing and route optimization using machine learning.
- **Pinterest:** Python is used to process and store huge amounts of image data efficiently.

Input and Output in Python

Understanding input and output operations is fundamental to Python programming. With the print() function, we can display output in various formats, while the input() function enables interaction with users by gathering input during program execution.

Taking input in Python

Python's input() function is used to take user input. By default, it returns the user input in form of a string.

Example:

```
name = input("Enter your name: ")

print("Hello,", name, "! Welcome!")
```

Output

```
Enter your name: GeeksforGeeks
Hello, GeeksforGeeks ! Welcome!
```

The code prompts the user to input their name, stores it in the variable "name" and then prints a greeting message addressing the user by their entered name.

To learn more about taking input, please refer: Taking Input in Python

Printing Output using print() in Python

At its core, printing output in Python is straightforward, thanks to the print() function. This function allows us to display text, variables and expressions on the console. Let's begin with the basic usage of the print() function:

In this example, "Hello, World!" is a string literal enclosed within double quotes. When executed, this statement will output the text to the console.

```
print("Hello, World!")
```

Output

```
Hello, World!
```

Printing Variables

We can use the print() function to print single and multiple variables. We can print multiple variables by separating them with commas. **Example:**

```
# Single variable
s = "Bob"
print(s)

# Multiple Variables
s = "Alice"
age = 25
city = "New York"
print(s, age, city)
```

Output

```
Bob
Alice 25 New York
```

Take Multiple Input in Python

We are taking multiple input from the user in a single line, splitting the values entered by the user into separate variables for each value using the split() method. Then, it prints the values with corresponding labels, either two or three, based on the number of inputs provided by the user.

```
# taking two inputs at a time
x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)

# taking three inputs at a time
x, y, z = input("Enter three values: ").split()
print("Total number of students: ", x)
print("Number of boys is : ", y)
```

print("Number of girls is : ", z)

Output

```
Enter two values: 5 10
Number of boys: 5
Number of girls: 10
Enter three values: 5 10 15
Total number of students: 5
Number of boys is : 10
Number of girls is : 15
```

How to Change the Type of Input in Python

By default input() function helps in taking user input as string. If any user wants to take input as int or float, we just need to typecast it.

Print Names in Python

The code prompts the user to input a string (the color of a rose), assigns it to the variable color and then prints the inputted color.

```
# Taking input as string

color = input("What color is rose?: ")

print(color)
```

Output

```
What color is rose?: Red
Red
```

Print Numbers in Python

The code prompts the user to input an integer representing the number of roses, converts the input to an integer using typecasting and then prints the integer value.

```
# Taking input as int

# Typecasting to int

n = int(input("How many roses?: "))

print(n)
```

Output

```
How many roses?: 88
88
```

Print Float/Decimal Number in Python

The code prompts the user to input the price of each rose as a floating-point number, converts the input to a float using typecasting and then prints the price.

```
# Taking input as float

# Typecasting to float

price = float(input("Price of each rose?: "))

print(price)
```

Output

```
Price of each rose?: 50.3050.3
50.3050.3
```

Find DataType of Input in Python

In the given example, we are printing the type of variable x. We will determine the type of an object in Python.

```
a = "Hello World"

b = 10

c = 11.22

d = ("Geeks", "for", "Geeks")

e = ["Geeks", "for", "Geeks"]

f = {"Geeks": 1, "for":2, "Geeks":3}

print(type(a))

print(type(b))

print(type(c))

print(type(d))

print(type(e))

print(type(f))
```

Output

```
<class 'str'>

<class 'int'>

<class 'float'>

<class 'tuple'>

<class 'list'>

<class 'dict'>
```

Python Variables

In Python, variables are used to store data that can be referenced and manipulated during program execution. A variable is essentially a name that is assigned to a value. Unlike many other programming languages, Python variables do not require explicit declaration of type. The type of the variable is inferred based on the value assigned.

Variables act as placeholders for data. They allow us to store and reuse values in our program.

Example:

```
# Variable 'x' stores the integer value 10

x = 5

# Variable 'name' stores the string "Samantha"

name = "Samantha"

print(x)

print(name)
```

Output

```
5

Samantha
```

Rules for Naming Variables

To use variables effectively, we must follow Python's naming rules:

- Variable names can only contain letters, digits and underscores (_).
- A variable name cannot start with a digit.
- Variable names are case-sensitive (myVar and myvar are different).
- Avoid using Python keywords (e.g., if, else, for) as variable names.

Valid Example:

```
age = 21

_colour = "lilac"

total_score = 90
```

Invalid Example:

```
1name = "Error"  # Starts with a digit

class = 10       # 'class' is a reserved keyword

user-name = "Doe"  # Contains a hyphen
```

Assigning Values to Variables

Basic Assignment

Variables in Python are assigned values using the = operator.

```
x = 5

y = 3.14

z = "Hi"
```

Dynamic Typing

Python variables are dynamically typed, meaning the same variable can hold different types of values during execution.

```
x = 10

x = "Now a string"
```

Multiple Assignments

Python allows multiple variables to be assigned values in a single line.

Assigning the Same Value

Python allows assigning the same value to multiple variables in a single line, which can be useful for initializing variables with the same value.

```
a = b = c = 100

print(a, b, c)
```

Output

```
100 100 100
```

Assigning Different Values

We can assign different values to multiple variables simultaneously, making the code concise and easier to read.

```
x, y, z = 1, 2.5, "Python"

print(x, y, z)
```

Output

```
1 2.5 Python
```

Type Casting a Variable

Type casting refers to the process of converting the value of one data type into another. Python provides several built-in functions to facilitate casting, including int(), float() and str() among others.

Basic Casting Functions

- **int()** - Converts compatible values to an integer.
- **float()** - Transforms values into floating-point numbers.
- **str()** - Converts any data type into a string.

Examples of Casting:

```
# Casting variables

s = "10"  # Initially a string

n = int(s)  # Cast string to integer

cnt = 5

f = float(cnt)  # Cast integer to float

age = 25

s2 = str(age)  # Cast integer to string


# Display results

print(n)

print(f)
print(s2)
```

Output

```
10

5.0

25
```

Getting the Type of Variable

In Python, we can determine the type of a variable using the type() function. This built-in function returns the type of the object passed to it.

Example Usage of type()

```
# Define variables with different data types

n = 42

f = 3.14

s = "Hello, World!"

li = [1, 2, 3]

d = {'key': 'value'}

bool = True


# Get and print the type of each variable

print(type(n))

print(type(f))

print(type(s))

print(type(li))

print(type(d))

print(type(bool))
```

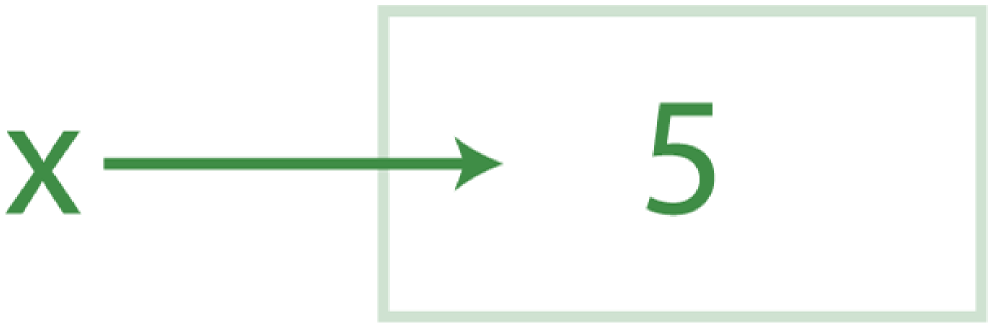
Output

```
<class 'int'>

<class 'float'>

<class 'str'>

<class 'list'>

<class 'dict'>

<class 'bool'>
```

Object Reference in Python

Let us assign a variable x to value 5.

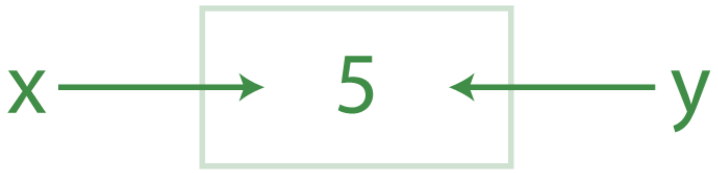
x = 5



When x = 5 is executed, Python creates an object to represent the value 5 and makes x reference this object.

Now, if we assign another variable **y** to the variable **x.**

y = x



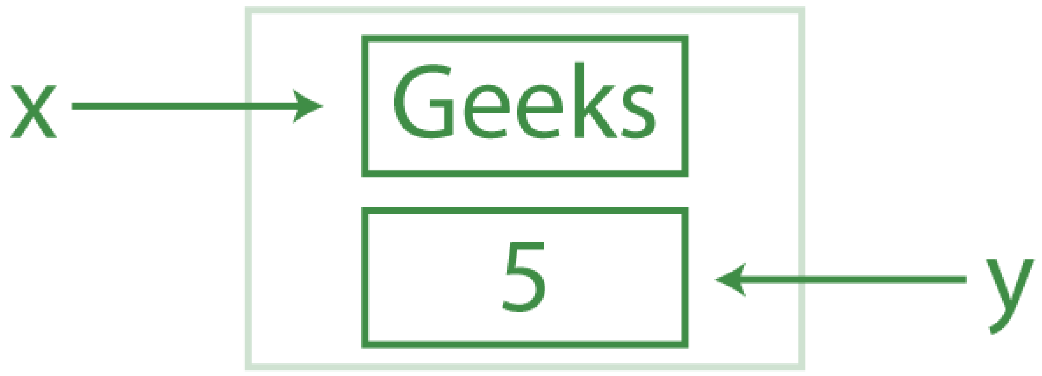
Explanation:

- Python encounters the first statement, it creates an object for the value 5 and makes x reference it. The second statement creates y and references the same object as x, not x itself. This is called a *Shared Reference*, where multiple variables reference the same object.

Now, if we write

x = 'Geeks'

Python creates a new object for the value "Geeks" and makes x reference this new object.

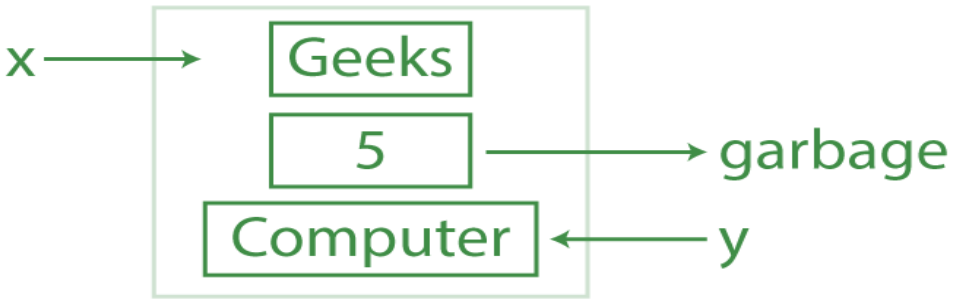


Explanation:

- The variable y remains unchanged, still referencing the original object 5.

If we now assign a new value to y:

y = "Computer"



- Python creates yet another object for "Computer" and updates y to reference it.
- The original object 5 no longer has any references and becomes eligible for garbage collection.

Key Takeaways:

- Python variables hold references to objects, not the actual objects themselves.
- Reassigning a variable does not affect other variables referencing the same object unless explicitly updated.

Delete a Variable Using del Keyword

We can remove a variable from the namespace using the del keyword. This effectively deletes the variable and frees up the memory it was using.

Example:

```
# Assigning value to variable

x = 10

print(x)

# Removing the variable using del

del x

# Trying to print x after deletion will raise an error

# print(x)  # Uncommenting this line will raise NameError: name 'x' is not defined
```

Explanation:

- del x removes the variable x from memory.
- After deletion, trying to access the variable x results in a NameError, indicating that the variable no longer exists.

Operators:

Python language supports the following types of operators.

Arithmetic Operators

Comparison (Relational) Operators

Assignment Operators

Logical Operators

Bitwise Operators

Membership Operators Identity Operators Let us have a look on all operators one by one. Python Arithmetic Operators Assume variable a holds 10 and variable b holds 20, then –

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) −	9//2 = 4 and 9.0//2.0 = 4.0, 11//3 = -4, 11.0//3 = -4.0

Python Keywords

Keywords in Python are special reserved words that are part of the language itself. They define the rules and structure of Python programs, which means you cannot use them as names for your variables, functions, classes, or any other identifiers.

Getting List of all Python keywords

We can also get all the keyword names using the below code.

```
import keyword


# printing all keywords at once using "kwlist()"

print("The list of keywords is : ")

print(keyword.kwlist)
```

Output

```
The list of keywords are:
['False', 'None', 'True',"__peg_parser__ 'and', 'as', 'assert', 'async', 'await', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

How to Identify Python Keywords ?

- **With Syntax Highlighting -** Most of IDEs provide syntax-highlight feature. You can see Keywords appearing in different color or style.
- **Look for SyntaxError -** This error will encounter if you have used any keyword incorrectly. Note that keywords can not be used as identifiers (variable or a function name).

What Happens if We Use Keywords as Variable Names ?

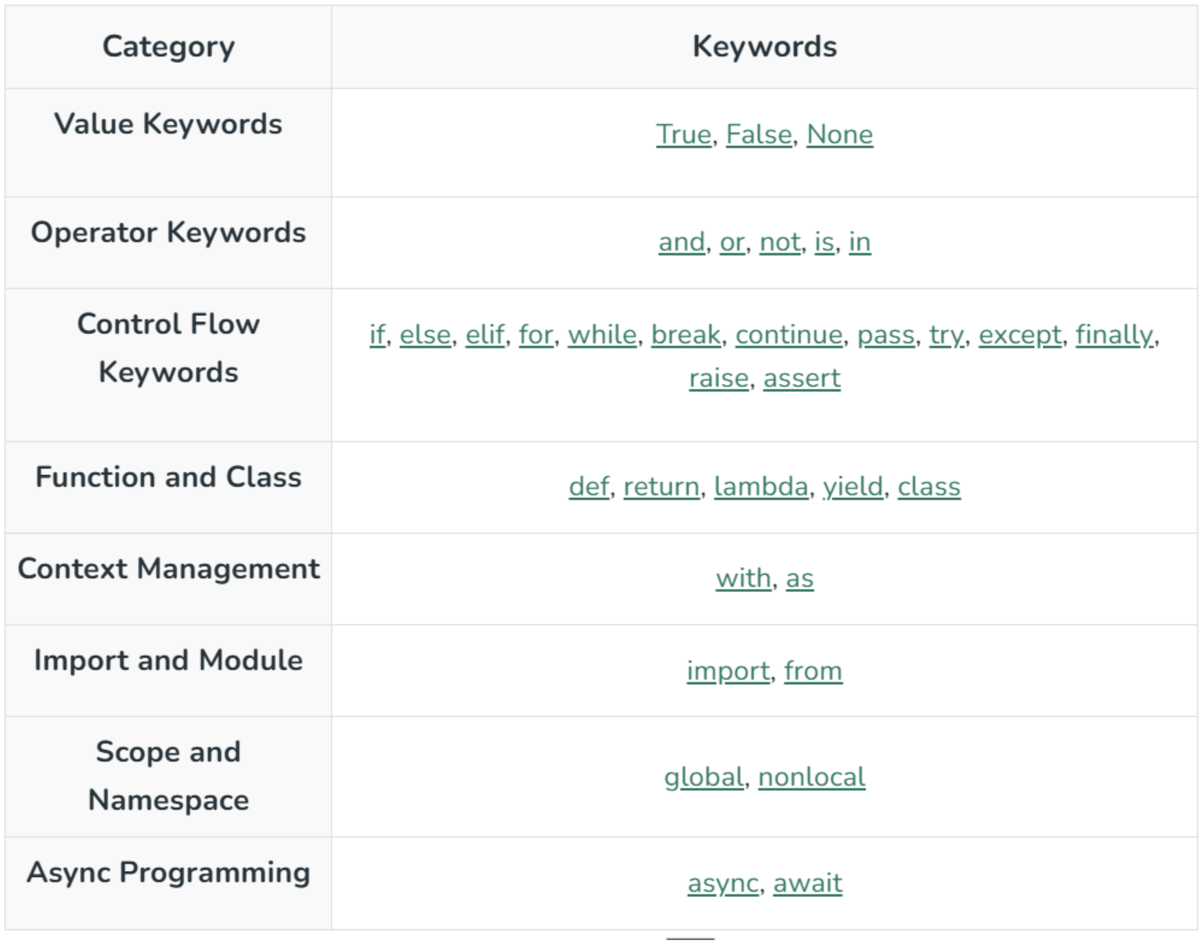
If we attempt to use a keyword as a variable, Python will raise a **SyntaxError.** Let's look at an example:

```
for = 10

print(for)
```

Output

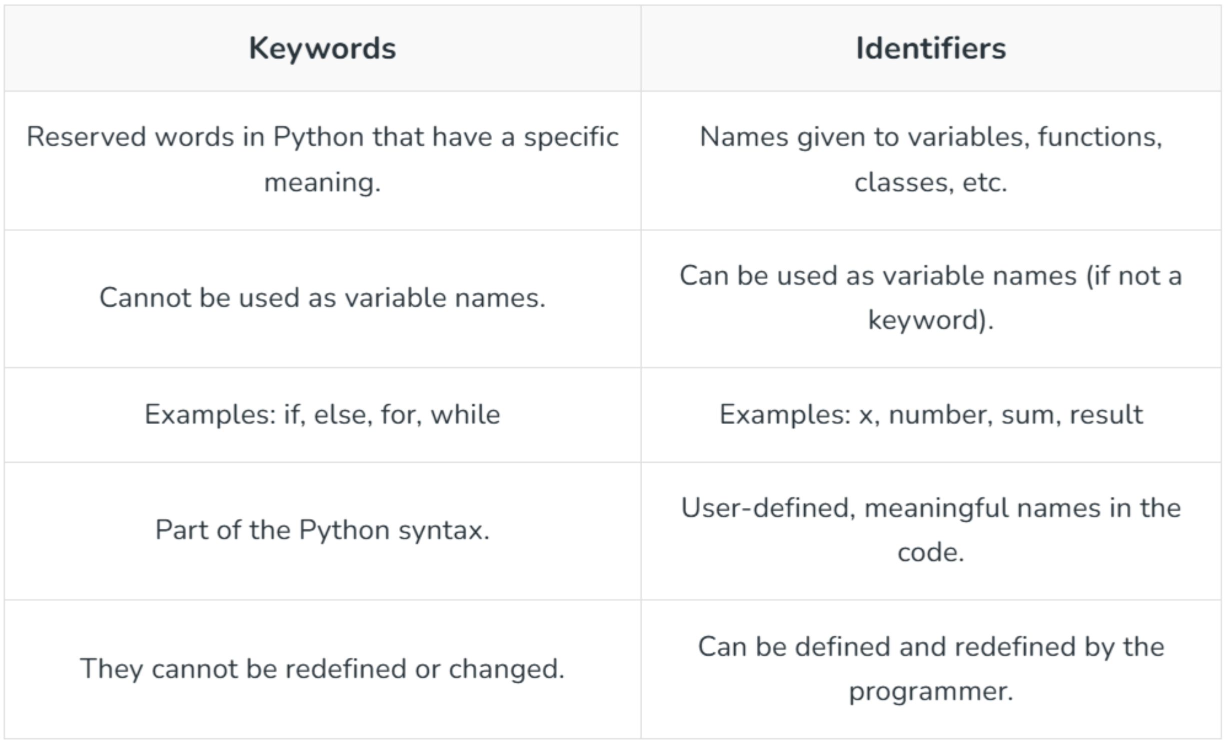
```
Hangup (SIGHUP)
File "/home/guest/sandbox/Solution.py", line 1
for = 10
^
SyntaxError: invalid syntax
```

Let's categorize all keywords based on context for a more clear understanding.



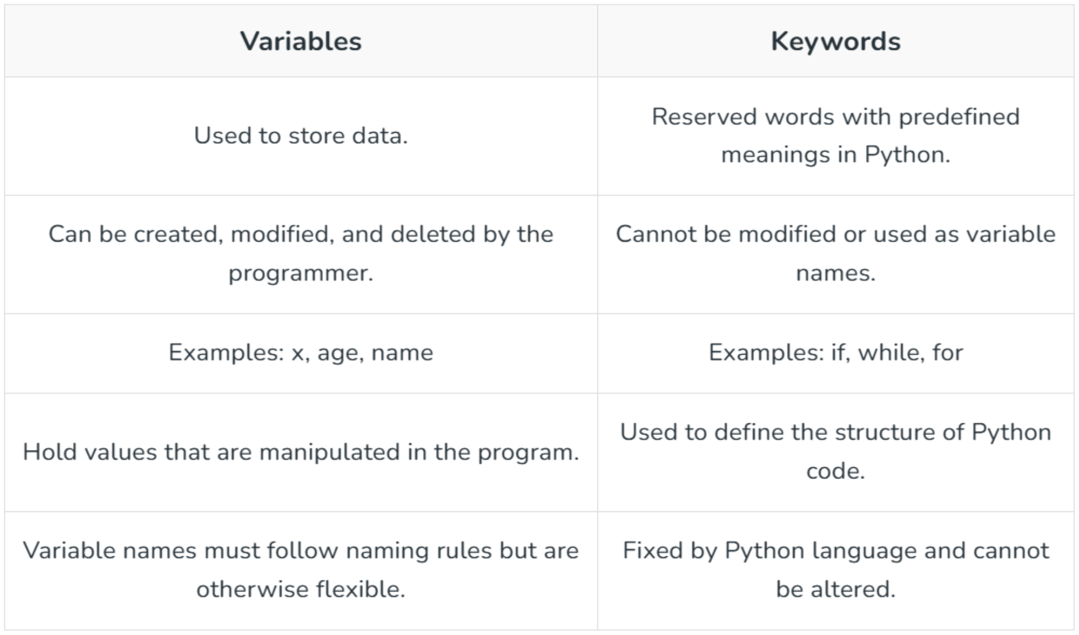
Category	Keywords
Value Keywords	True, False, None
Operator Keywords	and, or, not, is, in
Control Flow Keywords	if, else, elif, for, while, break, continue, pass, try, except, finally, raise, assert
Function and Class	def, return, lambda, yield, class
Context Management	with, as
Import and Module	import, from
Scope and Namespace	global, nonlocal
Async Programming	async, await

Keywords vs Identifiers



Keywords	Identifiers
Reserved words in Python that have a specific meaning.	Names given to variables, functions, classes, etc.
Cannot be used as variable names.	Can be used as variable names (if not a keyword).
Examples: if, else, for, while	Examples: x, number, sum, result
Part of the Python syntax.	User-defined, meaningful names in the code.
They cannot be redefined or changed.	Can be defined and redefined by the programmer.

Variables vs Keywords



Variables	Keywords
Used to store data.	Reserved words with predefined meanings in Python.
Can be created, modified, and deleted by the programmer.	Cannot be modified or used as variable names.
Examples: x, age, name	Examples: if, while, for
Hold values that are manipulated in the program.	Used to define the structure of Python code.
Variable names must follow naming rules but are otherwise flexible.	Fixed by Python language and cannot be altered.