

# Syntax

## Defination:

**Syntax** in a programming language is the set of rules that defines the structure and format of valid code statements and expressions, dictating how symbols, keywords, and operators can be combined to create meaningful programs.

## Why Python's Syntax is Easier Compared to Other Programming Languages

One of the key reasons why Python is so popular among beginners and experienced developers alike is its **easy-to-read syntax**. Let's compare how to print "Hello, World!" in Python with other programming languages like Java and C.

### Java

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

### Python

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

### C

```
#include <stdio.h>  
  
int main() {  
    printf("Hello, World!");  
    return 0;  
}
```

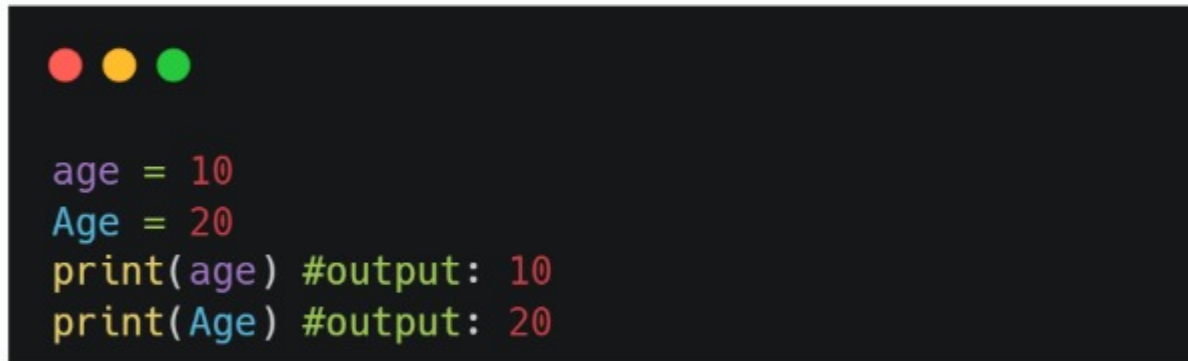
# Case Sensitivity:

## Explanation:

Case sensitivity means that Python treats uppercase and lowercase letters as different characters. So, Variable and variable are not the same, even though they look similar. Python distinguishes between them based on the case of the letters.

## Example:

You can give an example like this to demonstrate the concept:

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code that demonstrates case sensitivity by assigning values to variables 'age' and 'Age' and then printing them. The code shows that 'age' and 'Age' are treated as two different variables.

```
age = 10
Age = 20
print(age) #output: 10
print(Age) #output: 20
```

# Indentation

## Explanation:

In many programming languages, curly braces {} are used to define the code blocks, but Python uses indentation—spaces or tabs(1 tab = 4 spaces) at the beginning of a line to group related lines of code. Proper indentation is crucial; otherwise, Python will raise errors.

## Example:

### In C Programming:

with spaces:

```
if (a == 1) {  
    printf("This is Integer");  
}  
else {  
    printf("This is String");  
}
```

without spaces:

```
if (a == 1) {  
printf("This is Integer");  
}  
else {  
printf("This is String");  
}
```

### In Python Programming:

with spaces:

```
if a == 1:  
    print("This is Integer")  
else:  
    print("This is String")
```

without spaces:

```
if a == 1:  
    print("This is Integer")  
else:  
    print("This is String")
```

## Importance of Indentation:

**Code Readability:** Improves clarity and helps visualize the structure of code blocks, making it easier to understand the logic.

**Syntax Requirement in Python:** In Python, indentation is mandatory; incorrect indentation results in syntax errors that prevent the code from running.

**Error Prevention:** Proper indentation helps prevent logical errors by clearly defining control structures, reducing the likelihood of bugs in the code.



# Comments

## Defination:

Comments are non-executable lines in the code that are ignored by the interpreter or compiler. They are used to provide explanations or notes within the code.

## Purpose:

Emphasize that comments help make the code more understandable for anyone reading it, including the original author when returning to the code later.

## Types of Comments:

There are two types of comments: single-line comments and multi-line comments.

### Single-Line Comments:



```
# This is a single-line comment  
print("Hello, World!") # This prints a message
```

A single-line comment in Python starts with #, and everything after it on that line is ignored by the interpreter.

### Multi-Line Comments:



```
"""  
This is a multi-line comment.  
"""  
# Another way using single quotes  
'''  
This is also a multi-line comment.  
'''
```

A multi-line comment in Python is enclosed in triple quotes (""" or ''') and can span multiple lines, with all text inside ignored by the interpreter.



Example

# Output

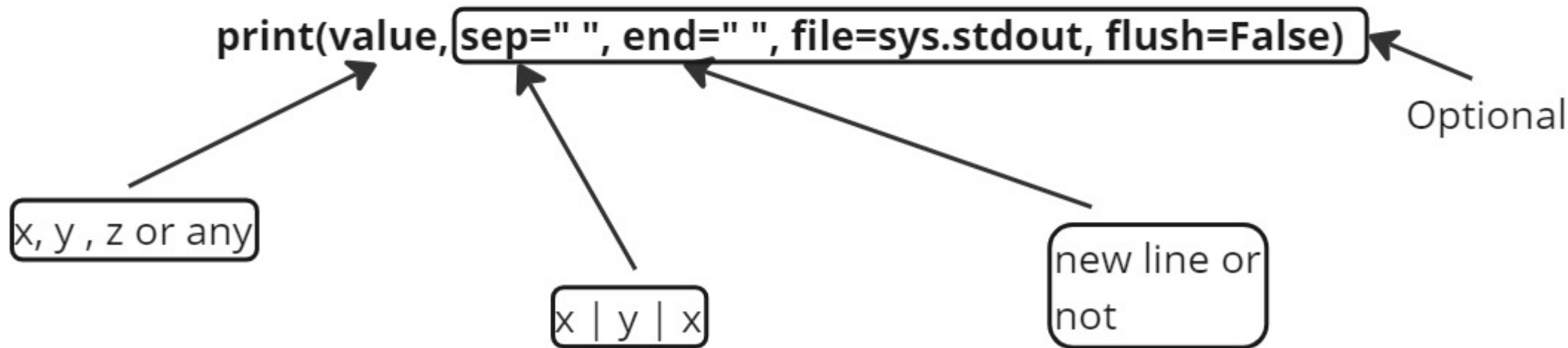
## print()

"The print() function in Python is used to display output. You can print text, numbers, or the result of expressions."



```
print("Hello, world!") # prints text
print(42) # prints a number
print(5 + 3) # prints the result of the expression (8)
```

The print() function in Python has several optional parameters that allow for customization of output. Here's a breakdown of the parameters you mentioned:



# Variable:

## Explanation:

A variable is a container for storing information that can be used later in a program. It can hold different types of data such as integers (int), decimals (float), or text (string)

## Purpose:

Variables allow programmers to store data that can be used later. Instead of hardcoding values into the code, variables let us assign and manage data dynamically, making the program more flexible and easier to maintain.

## Naming Variable:

When creating a variable, you give it a name that describes what it holds (e.g., age, total\_price, user\_name). This helps make the code more readable and understandable.

**Common Mistakes:** Mention some beginner mistakes, like forgetting quotes for string variables or using reserved keywords (e.g., print, for) as variable names.

There are different naming conventions like snake\_case, camelCase, and PascalCase.

### 1. Snake Case (snake\_case):

In **snake case**, all letters are lowercase, and words are separated by underscores (\_). Commonly used for variables and function names in Python.

```
user_name = "Alice"  
total_price = 150.50
```

### 2. Camel Case (camelCase):

In **camel case**, the first word starts with a lowercase letter, but every subsequent word starts with an uppercase letter. Common in JavaScript, but rare in Python.

```
userName = "Alice"  
totalPrice = 150.50
```

### 3. Pascal Case (PascalCase):

In **Pascal case**, the first letter of each word is capitalized, and no underscores or spaces are used. Typically used for class names.

```
class UserName:  
    pass
```

In Python, you can change a variable's type on the fly! One moment it can be a number, and the next it can hold a string. Python doesn't mind—it's super flexible like that!

# Variable Naming Rules:

A variable name must start with a letter or the underscore character

```
my_variable = 10 # Valid
_myVariable = 20 # Valid
2myVariable = 30 # Invalid (starts with a number)
```

**Rule:** Variable names must begin with either a letter (a-z, A-Z) or an underscore (\_).

A variable name cannot start with a number

```
age = 25 # Valid
user1_age = 30 # Valid
1st_user = "Alice" # Invalid (starts with a number)
```

**Rule:** Variable names cannot start with a digit.

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_)

```
score_2023 = 95 # Valid
totalAmount = 150.50 # Valid
total-amount = 200 # Invalid (contains a hyphen)
```

**Rule:** Variable names can only include letters, digits, and underscores. No special characters (like @, #, \$, etc.) are allowed.

Variable names are case-sensitive (age, Age and AGE are three different variables)

```
age = 25 # Valid
Age = 30 # Valid
AGE = 35 # Valid
print(age) # Outputs: 25
print(Age) # Outputs: 30
print(AGE) # Outputs: 35
```

**Rule:** Variable names are case-sensitive, meaning that age, Age, and AGE

Reserved keywords like if, else, for, and while cannot be used as variable names in Python.



# Data Types

A **data type** is a classification that specifies what kind of value a variable holds and what operations can be performed on that value in a programming language.

Python has the following data types built-in by default, in these categories:

Text Type: **str**

Numeric: **int, float, complex**

Sequence: **list, tuple, range**

Mapping: **dict**

Set Type: **set**

Boolean Type: **bool**

Binary Type: **bytes, bytearray, memoryview**

None Type: **NoneType**

## Input:

```
• • •
# Define variables with different data types
my_integer = 10      # int
my_float = 3.14      # float
my_string = "Hello"  # str
my_list = [1, 2, 3]  # list

# Print the data type of each variable
print("Data type of my_integer:", type(my_integer))
print("Data type of my_float:", type(my_float))
print("Data type of my_string:", type(my_string))
print("Data type of my_list:", type(my_list))
```

## Output:

```
• • •
Data type of my_integer: <class 'int'>
Data type of my_float: <class 'float'>
Data type of my_string: <class 'str'>
Data type of my_list: <class 'list'>
```

To print the data type of a variable in Python, you can use the **type()** function.

# String

In Python, a string is a sequence of characters enclosed in quotes, either single ( ' ') or double ( " ").

```
print("Hello")  
print('Hello')
```

## Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

```
a = "Hello"  
print(a)
```

## Quotes Inside Quotes

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
print("It's alright")  
print("He is called 'Johnny'")
```

## Multiline Strings

You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)
```

# String

## String Length

To get the length of a string, use the len() function.

```
a = "Hello, World!"  
print(len(a))
```

Python has a set of built-in methods that you can use on strings.

### Upper Case

The upper() method returns the string in upper case:

```
a = "Hello, World!"  
print(a.upper())
```

### Lower Case

The lower() method returns the string in lower case:

```
a = "Hello, World!"  
print(a.lower())
```

## F-Strings

F-String was introduced in Python 3.6, and is now the preferred way of formatting strings.

To specify a string as an f-string, simply put an f in front of the string literal, and add curly brackets {} as placeholders for variables and other operations.

```
age = 36  
txt = f"My name is John, I am {age}"  
print(txt)
```

## Int:

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
x = 1
y = 35656222554887711
z = -3255522
```

In Python, the int type has no maximum value and can grow as large as your machine's memory allows.

## Float:

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

```
x = 1.10
y = 1.0
z = -35.59
```

In Python, the float data type has a maximum finite value of approximately **1.7976931348623157e+308**. Beyond this value, any attempt to represent a larger number will result in inf (infinity).

## Bool:

Booleans represent truth values—either True or False.

```
is_logged_in = True
```

```
is_valid = (age > 18) # This will be True or False
```




# Input

## input()

The input() function in Python is used to take input from the user while a program is running. It allows you to ask the user for information and store their response for later use in your code. Here's how it works:

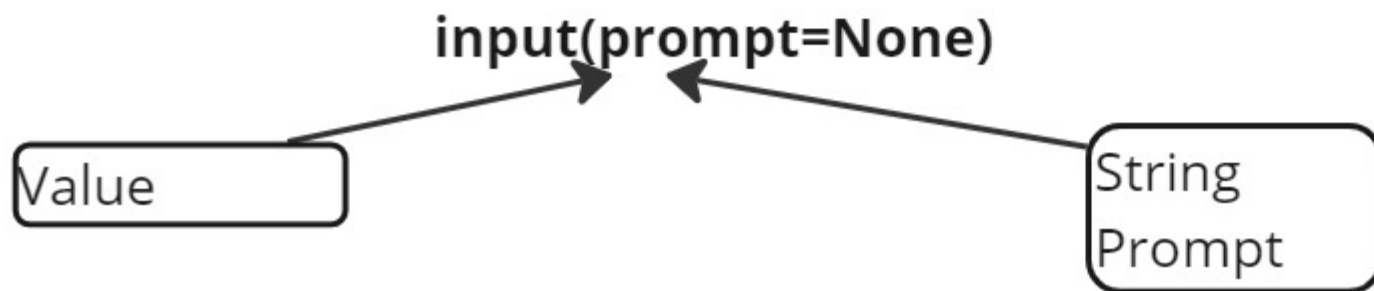
**Prompt for Input:** When the input() function is used, Python will display a message (if you provide one) and wait for the user to type something.

**User's Input:** Once the user types their input and presses **Enter**, the input is captured as a string (text) and stored in a variable for further processing.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains two lines of Python code: `name = input("Enter your name: ")` and `print("Hello, " + name + "!")`.

```
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

Here is the full syntax of input function



# Type Conversion

Type conversion, or type casting, allows you to change a variable's data type, which is useful when working with different data types or formatting output.

In Python, you can use built-in functions like `int()`, `float()`, and `str()` to convert variables to different types.

## Converting to Integer

You can convert a string that represents a whole number into an

```
string_number = "10" # String
integer_number = int(string_number) # Convert to integer
print(integer_number) # Output: 10
print(type(integer_number)) # Output: <class 'int'>
```

## Converting to Float

To convert a string representing a decimal number into a float, you can use the `float()` function.

```
number = 25 # Integer
string_number = str(number) # Convert to string
print(string_number) # Output: '25'
print(type(string_number)) # Output: <class 'str'>
```

## Converting to String

You can convert numbers to strings with the `str()` function, which is

```
string_float = "3.14" # String
float_number = float(string_float) # Convert to float
print(float_number) # Output: 3.14
print(type(float_number)) # Output: <class 'float'>
```

Type conversion is important for ensuring that your program runs correctly when dealing with different data types. By using `int()`, `float()`, and `str()`, you can easily convert between types to suit your needs.

Don't forget to **subscribe** for more tutorials.

# Thanks for Watching!

Hit the **like** button if this video helped you! forget to **subscribe** for more tutorials.

**Stay tuned for the next video!**