**Q.1 In the below elements which of them are values or an expression? Eg:- values can be integer or string and expressions will be mathematical operators.**

*

'hello'

-87.8

-

/

+

6

**Answer :**

| Element | Type | Description |
|---|---|---|
| * | Expression | Multiplication operator |
| 'Hello' | Value | String |
| -87.8 | Value | Floating-point number |
| - | Expression | Subtraction operator |
| / | Expression | Division operator |
| + | Expression | Addition operator |
| 6 | Value | Integer |

- Values : 'hello', -87.8, 6
- Expressions : *, -, /, +

# Q.2. What is the difference between string and variable?

# Answer :

The difference between String and Variable are as follows :

**1. String:**

A **string** is a sequence of characters enclosed in quotes. It is a data type used to represent text in programming.

Strings can contain letters, numbers, symbols and spaces.

**For Example :**

( "Hello, World!"

'12345'

"Python is fun!" )

In the example above, "Hello, World!", '12345' and "Python is fun!" are strings. They are literal values representing text.

**2. Variable:**

- A **variable** is a name or identifier used to store and refer to data in memory. It acts as a label for data.
- Variables can hold different data types, including strings, integers, floats, etc.

**For Example:**

( greeting = "Hello, World!"

number = 12345

is_fun = "Python is fun!" )

In the example above, greeting, number and is fun are variables. They are used to store the values "Hello, World!", 12345, and "Python is fun!" respectively.

**Key Points**

- **Strings** are actual data, whereas **variables** are labels or containers for storing that data.
- A string is a data type, while a variable is a named reference to data stored in memory.

**Example Use case :**

Here's a simple Python code snippet illustrating the difference between a **“string and a variable” :**

**# String**

"Hello, World!" [ # This is a string, but not stored in a variable ]

**# Variable**

message = "Hello, World!" [ # 'message' is a variable holding the string value ]

print(message) [ # Output: Hello, World! ]

**Code Explanation :**

- **"Hello, World!"** is a string that is directly used.
- **“message”** is a variable storing the string **"Hello, World!"** and it can be used to reference that string elsewhere in the code.

## Q.3. Describe three different data types ?

## Answer :

| Data Type | Description | Example Values | Common Operations |
|---|---|---|---|
| Integer | Whole numbers, no decimals | 5, -42, 1000 | Addition, Subtraction, Multiplication |
| Float | Real numbers with decimals | 3.14, -0.001, 2.5e6 | Division, Precision Calculations |
| String | Sequence of characters (text) | "hello", 'Python', "42" | Concatenation, Slicing, Formatting |

**1.Integer (int)**

**Definition:** Integers are whole numbers that can be positive, negative or zero. They do not have decimal points.

**Characteristics:**

Stored as binary numbers.

Used for counting and indexing operations.

Fixed memory allocation based on language or system architecture.

**Examples:**

**Positive Integers:** 5, 100, 42

**Negative Integers:** -3, -99, -500

**Zero:** 0

**Example with Code :**

pi = 3.14159 [ # A float representing the mathematical constant π ]

```
distance = 5.6        [ # A float representing distance ]
result = pi * distance   [ # Multiply pi with distance ]
print(result)             [ # Output: 17.592904 ]
```

**Here, pi and distance are floating-point numbers used to perform precision calculations**.

**2. Floating-Point Number (float)**

- Floating-point numbers represent real numbers and can contain fractional parts (decimals). They are used for precision calculations.

**Characteristics**:

- Include a decimal point.
- Stored in a format that can handle large ranges of values with a limited number of digits.
- Subject to precision limitations, leading to rounding errors.

**Examples**:

- **Positive Floats**: 3.14, 0.001, 100.5
- **Negative Floats**: -2.71, -0.0001, -75.75
- **Scientific Notation**: 1.23e4 (represents 12300)

**For Example** :

- Calculations involving fractions, scientific computations, and any scenario requiring precision.

**Example with Code** :

```
pi = 3.14159     [ # A float representing the mathematical constant π ]
distance = 5.6   [ # A float representing distance ]
result = pi * distance   [ # Multiply pi with distance ]
print(result)            [ # Output: 17.592904 ]
```

**Here, pi and distance are floating-point numbers used to perform precision calculations.**

**3. String (str)**

A string is a sequence of characters used to represent text. Strings can include letters, numbers, symbols, and spaces.

**Characteristics**:

- Enclosed in quotes (single ', double ", or triple """).
- Immutable in many programming languages, meaning they can't be changed once created.
- Support various operations like concatenation, slicing, and formatting.

**Examples**:

- **Simple Strings**: "hello", 'Python'
- **Multi-line Strings**: """This is a multi-line string"""
- **Special Characters**: "Hello\nWorld" (includes newline character)

**For Example :**

- Displaying messages, manipulating text, storing user input, and representing any non-numeric data.

**Example with Code** :

```
greeting = "Hello, World!"          [ # A string storing a greeting message ]
name = 'Alice'
message = greeting + " My name is " + name  [  # Concatenation of strings ]
print(message)                              [ # Output: Hello, World! My name is Alice ]
```

In this example, greeting, name and message are string variables that can be combined or manipulated to create more complex text.

**Key Points :**

- **Integers** are used for discrete values where precision is not a concern.
- **Floats** are used for continuous values where precision is required.
- **Strings** are used to represent textual data and can be manipulated in various ways to suit application needs.

# Q.4 What is an expression made up of? What do all expressions do?

# Answer :

An expression in programming and mathematics is a combination of variables, constants, operators, and functions that are evaluated to produce a value. Expressions can be as simple as a single constant or variable or as complex as a combination of multiple elements with various operations.

**Components of an Expression**

1. **Variables:** Placeholders that represent data values (e.g., x, y, price).
2. **Constants:** Fixed values (e.g., 5, 3.14, "hello").
3. **Operators:** Symbols that specify the type of operation to perform (e.g., +, -, *, /).
4. **Functions:** Procedures that perform operations and return a value (e.g., sin(x), max(a, b)).
5. **Parentheses:** Used to group parts of expressions and control the order of evaluation.

**Types of Expressions**

1. **Arithmetic Expressions:** Involve mathematical operations (e.g., 3 + 5, a * (b - c)).
2. **Relational Expressions:** Compare values and return a boolean result (e.g., x > 10, a == b).
3. **Logical Expressions:** Combine boolean values using logical operators (e.g., x < 5 && y > 10).
4. **String Expressions:** Involve string concatenation or manipulation (e.g., "Hello" + " World", length(name)).

**What All Expressions Do**

- **Evaluate to a Value:** Expressions are evaluated to produce a result. For example, the expression 3 + 5 evaluates to 8.
- **Control Flow:** Expressions can influence the flow of a program by providing conditions for branching (e.g., if (x > 10)).
- **Assign Values:** Expressions can be used to assign values to variables (e.g., result = 2 * 3).
- **Perform Calculations:** Expressions perform computations based on the operators and operands used (e.g., mathematical, logical).

**For Examples**

1. **Arithmetic Expression:**
    - **Expression:** 7 * (2 + 3)
    - **Evaluation:** First, evaluate the parentheses to get 2 + 3 = 5, then multiply by 7 to get 35.
2. **Relational Expression:**
    - **Expression:** x < 10
    - **Evaluation:** Compares the value of x with 10 and returns true or false.
3. **Logical Expression:**
    - **Expression:** (x > 5) && (y < 20)
    - **Evaluation:** Returns true if both conditions are true; otherwise, returns false.
4. **String Expression:**
    - **Expression:** "Hello " + "World"
    - **Evaluation:** Concatenates the two strings to produce "Hello World".

Understanding expressions and their evaluation is fundamental to programming and problem-solving, as they allow you to perform calculations, make decisions and manipulate data effectively.

## Q.5 This assignment statements, like spam = 10. What is the difference between an expression and a statement?

## Answer :

**1.Expressions**

An expression is a combination of variables, constants, and operators that is evaluated to produce a value.

- **Purpose:** Calculate or evaluate to a single value.
- **Examples:**
    - 5 + 3 (evaluates to 8)
    - x * y
    - a > b
    - "Hello" + "World"

**2.Statements**

A statement is an instruction that performs an action or command, such as declaring a variable or executing a loop.

- **Purpose:** Execute an action or sequence of actions.
- **Examples:**
    - spam = 10 (assignment statement)
    - if x > 5: (conditional statement)
    - for i in range(3): (loop statement)
    - print("Hello") (function call statement)

**Key Difference**

- **Expressions** evaluate to produce a value.
- **Statements** execute actions and may include expressions as part of their operation.

**Differences Between Expressions and Statements**

| Feature | Expression | Statement |
|---|---|---|
| **Purpose** | Evaluates to produce a value | Executes an action or command |
| **Result** | Produces a value | May not produce a value |
| **Use Case** | Calculations, evaluations, conditions | Assignments, loops, control flow, function calls |
| **Examples** | 5 + 3, x > 5, sqrt(4) | x = 10, if x > 5:, while True: |
| **Contains** | Can be part of a statement | Can contain expressions |

**For Example with Code :**

**[ spam = 10 + 5 ]**

- **Expression:** 10 + 5 (evaluates to 15)
- **Statement:** spam = 10 + 5 (assigns 15 to spam)

In short, expressions focus on calculating values, while statements focus on executing actions.

**Q.6 After running the following code, what does the variable bacon contain?**

**bacon = 22**

**bacon + 1**

**Answer :**

To understand better way what the variable bacon contains lets understand with code.

**Code :**

```
bacon = 22          [ # Line 1: Assigns the value 22 to the variable bacon ]
bacon + 1           [ # Line 2: Evaluates to 23 but does not change the value of bacon ]
```

**Explanation of Code :**

1. **Line 1:**
    - bacon = 22 assigns the integer value 22 to the variable bacon.
2. **Line 2:**
    - bacon + 1 is an expression that evaluates to 23 because it adds 1 to the current value of bacon, which is 22.
    - However, **this line does not modify the variable bacon** itself. It simply evaluates the expression and produces a result, but the result is not stored or assigned back to bacon.

After running both lines of code, the variable bacon still contains the **value 22.** The expression **bacon + 1 does** not alter the original value of bacon because there is no assignment operation that updates bacon with the new value.

**Solution**

To actually change the value of bacon, you would need to update it explicitly with an assignment, like this:

**bacon = 22 [ # Initialize bacon with 22 ]**

**bacon = bacon + 1 [ # Assign the new value back to bacon ]**

**Or using the shorthand increment operator:**

**bacon = 22 [ # Initialize bacon with 22 ]**

**bacon += 1 [ # Shorthand for bacon = bacon + 1 ]**

**In both of these cases :**

- bacon would then contain **23.**
- The variable bacon contains **22** after running the code.

**Q.7 What should the values of the following two terms be?**

**1.'spam' + 'spamspam'**

**2.'spam' * 3**

**Answer :**

Let's understand the two terms to determine their values :

**1. 'spam' + 'Spamspam'**

- **Operation:** This expression uses the + operator to concatenate two strings: 'spam' and 'spamspam'.
- **Result:** 'spam' + 'spamspam' results in 'spamspamspam'.

**Explanation:**

- The + operator in Python, when used with strings, performs string concatenation. It joins the two strings together without any additional characters.
- So, 'spam' combined with 'spamspam' produces the single string 'spamspamspam'.

**2. 'spam' * 3**

- **Operation:** This expression uses the * operator to repeat the string 'spam' three times.
- **Result:** 'spam' * 3 results in 'spamspamspam'.

**Explanation:**

- The * operator in Python, when used with a string and an integer, repeats the string the specified number of times.
- Here, 'spam' is repeated 3 times, producing the string 'spamspamspam'.

Both expressions result in the same string, 'spamspamspam', but they achieve this through different operations:

| Expression | Operation | Result |
|---|---|---|
| 'spam' + 'spamspam' | Concatenation | 'spamspamspam' |
| 'spam' * 3 | Repetition | 'spamspamspam' |

- **'spam' + 'spamspam'** evaluates to **'spamspamspam'**
- **'spam' * 3** evaluates to **'spamspamspam'**

## Q.8 Why is eggs a valid variable name while 100 is invalid?

## Answer :

In Python and most programming languages, variable names must follow certain rules to be considered valid. Understanding these rules helps ensure that our code is syntactically correct and readable.

Let's look at why eggs is a valid variable name while 100 is not:

**( A) Why eggs is a Valid Variable Name**

- **Starts with a Letter:** eggs begins with a letter, which is allowed in Python variable names.
- **Consists of Letters and Digits:** Variable names can consist of letters (uppercase and lowercase), digits, and underscores. eggs consists solely of letters.
- **Follows Naming Conventions:** It adheres to standard naming conventions, which include using lowercase letters for variables unless specific naming conventions are needed.

**( B) Why 100 is an Invalid Variable Name**

- **Starts with a Digit:** Variable names cannot start with a digit. Python enforces this rule to distinguish variable names from numeric literals.
- **Misinterpreted as a Number:** A variable name like 100 would be ambiguous as it could be confused with the number 100.

**Python's Variable Naming Rules**

To further illustrate, here are the key rules for naming variables in Python:

1. **Start with a Letter or Underscore:**
    - Variable names must begin with a letter (a-z, A-Z) or an underscore (_).
    - Examples: myVariable, _privateVar, counter1
2. **Use Letters, Digits, or Underscores:**
    - The rest of the variable name can include letters, digits (0-9), or underscores.
    - Examples: var_123, data_point, resultValue
3. **Avoid Starting with a Digit:**
    - Names starting with digits are not allowed, as they would conflict with numeric literals.
    - Invalid: 123variable

4. **No Spaces or Special Characters:**
    - Variable names cannot include spaces or special characters like @, #, !, etc.
    - Use underscores instead of spaces: my_variable_name
5. **Case Sensitivity:**
    - Python is case-sensitive, so myVar, MyVar, and myvar are considered different variables.
    - Be consistent in naming conventions to avoid confusion.
6. **Avoid Reserved Keywords:**
    - Variable names should not be the same as Python's reserved keywords, such as for, if, else, class, def, etc.
    - Attempting to use a keyword will result in a syntax error.

**Examples of Valid and Invalid Variable Names are as follows :**

| Valid Variable Names | Invalid Variable Names |
|---|---|
| eggs | 100 |
| spam_eggs | 1_variable |
| _privateVar | @home |
| variable123 | my-variable |
| data_point | class (reserved keyword) |
| resultValue | total$ |

- **eggs** is a valid variable name because it follows all the naming rules: it starts with a letter, contains only letters, and is not a reserved keyword.
- **100** is an invalid variable name because it starts with a digit, which violates the naming convention rules in Python. Starting with a digit would make it indistinguishable from numeric values in the code.

Using valid **variable names** ensures that our code is syntactically correct and easy to understand, while adhering to **Python's naming conventions** helps maintain consistency and readability in our programming projects.

## Q.9 What three functions can be used to get the integer, floating-point number, or string version of a value?

## Answer :

In Python, we can convert values between different data types using specific built-in functions. The three primary functions used to convert a value to an integer, floating-point number or string are as follows :

**1. int() Function**

- **Purpose:** Converts a value to an integer.
- **Usage:** You can use int() to convert a string or a floating-point number to an integer.
- **Syntax:** int(value)

**Examples with code :**

```
int("42")        [ # Converts string to integer: 42 ]
int(3.14)      [ # Converts float to integer by truncation: 3 ]
int(True)      [ # Converts boolean to integer: 1 ]
int(False)     [ # Converts boolean to integer: 0 ]
```

**Notes:**

- When converting a float, the decimal part is truncated, not rounded.
- If you convert a string to an integer, the string must represent a valid integer value; otherwise, it will raise a ValueError.
- You can also use int() to convert from different bases (e.g., binary, octal, hexadecimal) by specifying the base as the second argument:

```
int("1010", 2)          [  # Converts binary string to integer: 10 ]
int("1A", 16)           [ # Converts hexadecimal string to integer: 26 ]
```

**2. float() Function**

- **Purpose:** Converts a value to a floating-point number.
- **Usage:** You can use float() to convert a string or an integer to a float.
- **Syntax:** float(value)

**For Examples with code :**

```
float("3.14")          [ # Converts string to float: 3.14   ]
float(42)              [  # Converts integer to float: 42.0 ]
float("2.718")         [ # Converts string to float: 2.718 ]
```

**Notes:**

- The string must be a valid representation of a floating-point number; otherwise, it will raise a ValueError.
- Floats can represent decimal numbers with more precision than integers.

**3. str() Function**

- **Purpose:** Converts a value to a string.
- **Usage:** You can use str() to convert any data type to a string.
- **Syntax:** str(value)

**For Examples with code :**

```
str(42)          [ # Converts integer to string: "42" ]
str(3.14)        [ # Converts float to string: "3.14" ]
str(True)        [ # Converts boolean to string: "True" ]
str(None)        [   # Converts NoneType to string: "None" ]
```

**Notes:**

- The str() function is very versatile and can convert virtually any object to its string representation.
- Useful for formatting output, concatenating with other strings, and logging or displaying data.

**For Example :**

Below is the example that demonstrates how these functions can be used in a practical scenario:

```
# Original values
integer_value = 42
float_value = 3.14159
string_value = "100"

# Convert to different types
converted_to_int = int(float_value)      # 3 (truncated)
converted_to_float = float(string_value) # 100.0
converted_to_str = str(integer_value)    # "42"

# Display the results
print("Integer:", converted_to_int)      # Output: Integer: 3
print("Float:", converted_to_float)      # Output: Float: 100.0
print("String:", converted_to_str)       # Output: String: "42"
```

- **int():** Converts to an integer.
- **float():** Converts to a floating-point number.
- **str():** Converts to a string.

## Q.10 Why does this expression cause an error? How can you fix it?

## 'I have eaten ' + 99 + ' burritos.'

## Answer :

The expression **[ 'I have eaten ' + 99 + ' burritos.' ]** causes an error in Python because it attempts to concatenate a string with an integer directly.

Let's understand the issue and how we can fix it.

**Why the Error Occurs**

In Python, you can only concatenate strings with other strings using the + operator. In this expression:

**Below is the code 1 :**

**[ 'I have eaten ' + 99 + ' burritos.' ]**

- **'I have eaten '** is a string.
- **99** is an integer.
- **' burritos.'** is a string.

When Python tries to execute the concatenation, it encounters 99, an integer, between two strings. This causes a **TypeError** because Python does not automatically convert the integer 99 into a string for concatenation purposes.

**Error Message**

The specific error message below one we might see is: **Code : 2**

**[ TypeError: can only concatenate str (not "int") to str ]**

**How to Fix It**

To fix this issue, we need to convert the integer 99 into a string before concatenating it with the other strings. You can achieve this using the str() function, which converts any given value into its string representation.

**Corrected Expression**

Here's how we can correct the expression:

**[ 'I have eaten ' + str(99) + ' burritos.' ]**

**Explanation of the Fix**

- str(99) converts the integer 99 to a string '99'.
- Concatenation: After conversion, you can concatenate 'I have eaten ', '99', and ' burritos.' using the + operator without any issues.
- The corrected expression will produce a single string: 'I have eaten 99 burritos.'.

**Complete Example**

**Here's below is the complete example that demonstrates the corrected expression: Code : 3**

- # Original expression (causes error)
- # print('I have eaten ' + 99 + ' burritos.') # This line would cause a TypeError
- # Corrected expression

**[ print('I have eaten ' + str(99) + ' burritos.') ]**

**Output : code : 4**

The corrected code will output:

**[ I have eaten 99 burritos. ]**

**Alternative Solution: Using f-strings Code : 5**

In Python 3.6 and later, we can also use f-strings for string formatting, which is a more modern and readable approach:

**[ print(f'I have eaten {99} burritos.') ]**

- f'I have eaten {99} burritos.' automatically converts the integer 99 to a string within the curly braces {}.

- Output: The same as before, it will print: **Code 6**

**[ I have eaten 99 burritos. ]**

**Conclusion :**

- **Error Cause:** The attempt to concatenate an integer with strings directly causes a TypeError.
- **Solution:** Convert the integer to a string using str() before concatenation or use f-strings for a cleaner approach.
- By addressing this type mismatch, the expression can be successfully evaluated without any errors.