Assignment - 1

1. In the below elements which of them are values or an expression?

- , 'hello', -87.8, - , / , +, 6

`Ans` : Here values are : `'hello'` , `-87.8` , `6` and Expressions are : `*` , `-` , `/` , `+`

2. What is the difference between string and variable?

`Ans` :

- A **variable** is like a container that can store values. It represents a memory location where data can be stored. The main purpose of variable is to store the data from one part of the program so that, it can be used in the other part of the program. Variables can store various types of information, such as numbers, text, or other data.
- A **string** is a specific type of information that can store in a variable. Strings represent text and are usually enclosed in double quotes (") or in single quotes (').
- e.g: **greet = "Hello, world!"**
  - here `greet` is a *variable* and `"Hello, world!"` is its *string* value

3. Describe three different data types.

`Ans` : Three different data types are -

1. **Integer** : It contains positive or negative whole numbers (without fractions or decimals). This is represented by `int` class. e.g: 4, -56, 0 etc...
2. **String** : It is a collection of one or more characters put in a single quote or double-quote. It generally represents textual data and class type is `str` . e.g: "hello", 'ten', "anything" etc...
3. **Floating Point** : It is a real number with a floating-point representation. It is specified by a decimal point. This is represented by the `float` class. e.g: 4.23, 789.1 etc...

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

`Ans` : An expression is a combination of operators, operands, constants and variables that is interpreted to produce some other value. An expression may consist of one or more operands, and zero or more operators to produce a value.

- For example, **Result = a + b * c** is an expression. Here, `Result` is a variable which stores the expression value, `a` , `b` , `c` are operands, and `+` , `*` are operators.

Different types of expressions are -

- **Constant expressions**: Constant Expressions consists of only constant values. A constant value is one that doesn’t change.
  - e.g: 5, 10 + 5 / 6.0, 'x’
- **Arithmetic Expressions**: An arithmetic expression is a combination of numeric values, operators, and sometimes parenthesis. The result of this type of expression is also a numeric value.
  - e.g: x = 2, y = 7 then (x+y)*2 = 18
- **Integral Expressions**: These are the kind of expressions that produce only integer results after all computations and type conversions.
  - e.g: a=6, b=3 then a+b-5 = 4
- **Floating Expressions**: These are the kind of expressions which produce floating point numbers as result after all computations and type conversions.
  - e.g: l=3, m=2 then l/m = 1.5
- **Relational Expressions**: In these types of expressions, arithmetic expressions are written on both sides of relational operator (> , < , >= , <=). Those arithmetic expressions are evaluated first, and then compared as per relational operator and produce a boolean output in the end. These expressions are also called Boolean expressions.
  - e.g: x <= y, x + y > 2
- **Logical expressions**: Logical Expressions combine two or more relational expressions and produces bool type results.
  - e.g: x > y && x == 10, x == 10 || y == 5
- **Bitwise Expressions**: These are the kind of expressions in which computations are performed at bit level.
  - e.g: x << 3 --> shifts three bit position to left
- **Combinational Expressions**: We can also use different types of expressions in a single expression, and that will be termed as combinational expressions.
  - e.g: c = a + (b >> 1)

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

`Ans` :

- An **expression** is any code that evaluates to a value. It can be a combination of variables, literals, operators, operands, and function calls. Expressions can be assigned to variables or used as operands in other expressions.
  - e.g: 5 + 3 (evaluates to 8), x * y (where x and y are variables), Math.sqrt(9) (a function call that evaluates to 3), etc.
- A **statement** is a group of expressions, or other statements, or combination of both, which are designed to carry out a task or action. Statements don’t necessarily evaluate to a value, but they create side effects or perform actions. Statements cannot be directly assigned to variables or used as operands.
  - e.g: Assignment statements like spam = 10, Control flow statements (if, while, for), Function and class declarations, etc.

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

bacon = 22 bacon + 1

`Ans` : Firstly, the number 22 is assigned to a varibale called 'bacon'. Then an expression 'bacon + 1' is evaluated. But after running both of these code the variable 'bacon' is still having the same value i.e 22 because here we did not change the variable assignment.

In [5]:
```
bacon = 22
bacon + 1
print(bacon)
```

```
22
```

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

'spam' + 'spamspam' 'spam' * 3

`Ans` :

- When `+` operator is used between two more strings, those strings will get concatenated. That means, those multiple strings are combined into a single string.
  - Hence in this case, `'spam' + 'spamspam'` results the combined string `'spamspamspam'` .
- When we try to multiply a string by an integer using the `*` operator, then the string will get repeated that many times.
  - So, here `'spam' * 3` gives us the three times repeated string `'spamspamspam'` .
- Hence, here both the expressions are returning the same output.

In [8]:
```
print('spam' + 'spamspam')
print('spam' * 3)
```

```
spamspamspam
spamspamspam
```

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

`Ans` : In Python, variable names must be adhere to some specific rules. These rules ensure consistency, readability, and compatibility with the Python's programming syntax.

- A valid variable name:
  - Starts with a letter (a-z or A-Z) or an underscore (_).
  - Cannot begin with a number.
  - Can only contain alphanumeric characters (letters and digits) and underscores (A-z, 0-9, and _).
  - Cannot have spaces in between.
  - Is case-sensitive (e.g. age and Age are two different variables).
  - Cannot be a Python keyword (reserved word).
- Hence, here `eggs` is a valid variable name while `100` is invalid.

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

`Ans` : In Python, a value can be converted from one object data type to a different object data type. This process is known as Type Casting. Python has various in-built functions to do type casting.

- To convert a value to an integer, `int()` function can be used. (Note: 'string' cannot be converted into 'int')
  - e.g: int(5.67) returns 5, int(False) will returns 0, etc.
- To convert a value to a floating-point number, we can use `float()` function. (Note: 'string' cannot be converted into 'float')
  - e.g: float(8) results 5.0, float(True) will return 1.0, etc.
- To convert a value to a string, we use `str()` function. It converts other data types (such as integers or floats) into strings.
  - e.g: str(12) will give '12', str(7.42) will results '7.42', str(True) returns 'True', etc.

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

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

`Ans` : In Python, we cannot concatenate string values with numbers. In the above expression, we are trying to concatenate two string values `'I have eaten '` and `' burritos.'` with an integer value `99` but it will throw an error.

```
In [23]: 'I have eaten ' + 99 + ' burritos.'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 'I have eaten ' + 99 + ' burritos.'

TypeError: can only concatenate str (not "int") to str
```

To fix this, we can convert the integer to a string using the `str()` function.

```
In [25]: 'I have eaten ' + str(99) + ' burritos.'   # number 99 is converted to string.

Out[25]: 'I have eaten 99 burritos.'
```

```
In [ ]:
```