1. **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**

**-**

**/**

**+**

The elements you’ve provided can be categorized as follows:

* **Values**:
  + 'hello': This is a string value.
  + -87.8: This is a floating-point number value.
* **Expressions**:
  + \*, -, /, +: These are mathematical operators, which are used in expressions to perform operations on values.

1. **What is the difference between string and variable?**

In Python, a variable and a string are two different concepts:

**Variable**: A variable is a named location in memory where a programmer can store data and later retrieve the data using the variable name. In Python, variables do not need explicit declaration to reserve memory space. The declaration happens automatically when a value is assigned to a variable. The equal sign (=) is used to assign values to variables.

x = 10

name = "Alice"

Here, x and name are variables. x holds the integer value 10 and name holds the string value "Alice".

**String:** A string in Python is a sequence of characters. Strings in Python are immutable. We can verify this by trying to change a part of a string which leads to an error. Strings can be declared using single quotes or double quotes.

s = "Hello, World!"

Here, s is a variable that holds the string value "Hello, World!".

So, the difference is that a variable is a container for storing data, and a string is a type of data that can be stored in a variable. A variable can hold various types of data like integers, floats, strings, lists, etc.

1. **Describe three different data types.**

**Integer (int)**: This data type represents whole numbers, both positive and negative. For example, 5, -3, 0 are all integers.

**Float (float**): This data type represents real numbers, i.e., numbers with a decimal point. For example, 3.14, -0.01, 5.0 are all floats.

**String (str):** This data type represents a sequence of characters enclosed within single quotes (' ') or double quotes (" "). For example, 'Hello, World!', "Python" are strings.

Here’s how you can define these data types in Python:

# Integer

x = 10

print(type(x)) # Output: <class 'int'>

# Float

y = 3.14

print(type(y)) # Output: <class 'float'>

# String

z = "Hello, Python!"

print(type(z)) # Output: <class 'str'>

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

An **expression** is a combination of **operators**, **constants**, and **variables** that evaluates to a single value. Let’s break it down:

1. **Operators**: These are symbols or functions that perform specific operations on operands. Examples include addition (+), subtraction (-), multiplication (\*), and division (/). Operators dictate how the values in an expression interact with each other.
2. **Constants**: Constant expressions consist of only fixed values that do not change. For instance, 5, 10 + 5 / 6.0, and 'x' are constant expressions.
3. **Variables**: Variables represent placeholders for values. They can change during program execution. For example, if x and y are integer variables, expressions like x, x \* y, and x + int(5.0) fall into this category.

Now, let’s explore the different types of expressions:

* **Constant Expressions**: These consist solely of constant values. A constant value remains unchanged. Examples include 5, 10 + 5 / 6.0, and 'x'.
* **Integral Expressions**: These produce integer results after considering automatic and explicit type conversions. For instance, expressions like x, x \* y, and x + int(5.0) fall into this category.
* **Float Expressions**: These yield floating-point results after type conversions. Examples include x + y and 10.75, where x and y are floating-point variables.
* **Relational Expressions**: These evaluate to Boolean values (true or false). When arithmetic expressions appear on either side of a relational operator, they are first evaluated, and then the results are compared. Examples include x <= y and x + y > 2.
* **Logical Expressions**: These combine multiple relational expressions and produce Boolean results. For instance, x > y && x == 10 and x == 10 || y == 5.
* **Pointer Expressions**: These produce address values. Examples include &x, ptr, and ptr++, where x is a variable and ptr is a pointer.
* **Bitwise Expressions**: These manipulate data at the bit level. They are used for testing or shifting bits. Examples include x << 3 (shifts three positions left) and y >> 1 (shifts one position right). Shift operators are often used for multiplication and division by powers of two.

Remember, expressions can also be combinations of the above types, known as **compound expressions**

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

In Python, an **expression** and a **statement** are two different concepts:

1. **Expression**: An expression is a piece of code that produces a value when evaluated. It consists of values combined together with operators. Expressions need to be evaluated. For example, 2 + 3 is an expression, and when it is evaluated, it produces the value 5.
2. **Statement**: A statement is an instruction that Python can execute. Assignment statements like spam = 10, conditional statements like if, for and while loops are all examples of statements. Statements do something and don’t have a value.

Here’s a simple way to distinguish between the two: If you can print it, or assign it to a variable, it’s an expression. If you can’t, it’s a statement.

Here’s a Python code snippet that demonstrates this:

# Expression

result = 2 + 3

print(result) # Output: 5

# Statement

spam = 10 # This is a statement, not an expression.

print(spam) # Output: 10

In the above code, 2 + 3 is an expression which gets evaluated to 5, and spam = 10 is a statement which assigns the value 10 to the variable spam. The print() function is also a statement which performs an action (printing to the console).

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

**bacon = 22**

**bacon + 1**

The variable bacon will still contain the value 22. Here’s why:

In the code you provided:

bacon = 22

bacon + 1

The second line (bacon + 1) does increase the value of bacon by 1, but this new value is not stored anywhere. Therefore, the value of bacon remains 22.

If you want to increase the value of bacon by 1 and save it, you should use the += operator like this:

bacon = 22

bacon += 1

Now, bacon would contain the value 23.

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

**'spam' + 'spamspam'**

**'spam' \* 3**

In Python, the + operator concatenates strings, and the \* operator repeats a string a given number of times. So, the values of the two terms would be:

* 'spam' + 'spamspam' results in 'spamspamspam'
* 'spam' \* 3 also results in 'spamspamspam'

So, both expressions give the same result: 'spamspamspam'.

1. **Why is eggs a valid variable name while 100 is invalid?**

In Python, variable names must adhere to certain rules to be considered valid. Let’s break down why **“eggs”** is a valid variable name while **“100”** is invalid:

1. **Starting Character**:
   * A valid variable name must start with a **letter** (either uppercase or lowercase) or an **underscore (\_)**.
   * **“eggs”** starts with a letter (‘e’), making it valid.
   * **“100”** starts with a digit (‘1’), which is not allowed.
2. **Subsequent Characters**:
   * After the initial character, a variable name can include letters, digits, and underscores.
   * **“eggs”** contains only letters, so it satisfies this rule.
   * **“100”** consists entirely of digits, violating this rule.
3. **Reserved Words**:
   * Certain words are reserved in Python (e.g., **“if,” “else,” “while,” “for,”** etc.). These words cannot be used as variable names.
   * **“eggs”** is not a reserved word, so it’s acceptable.
   * **“100”** is not a reserved word either, but it starts with a digit, which makes it invalid.

In summary, variable names like **“eggs”** are valid because they follow the rules, while **“100”** violates the rules by starting with a digit. Remember that meaningful and descriptive variable names enhance code readability!

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

In Python, you can use the following three functions to manipulate data types:

1. **int()**: This function converts a value to an integer. It’s useful when you want to work with whole numbers, both positive and negative, without decimals.
2. **float()**: Use this function to convert a value to a floating-point number. Floating-point numbers include decimals and are suitable for more precise calculations.
3. **str()**: When you need to represent a value as a string (a sequence of characters), the str() function comes in handy. It converts any data type to its string representation.

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

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

The expression 'I have eaten ' + 99 + ' burritos.' results in an error because it attempts to concatenate a string ('I have eaten ') with an integer (99) and another string (' burritos.'). In programming, concatenation is only valid between strings.

To fix this error and achieve the desired output of 'I have eaten 99 burritos.', you can convert the integer 99 into a string by enclosing it in single or double quotes. Here’s the corrected expression:

'I have eaten ' + str(99) + ' burritos.'

In this modified version, the str(99) converts the integer 99 to a string, allowing successful concatenation with the other strings.