Data Types

In Python, data types refer to the kind of values that can be stored in a variable.
Python has several built-in data types, including:
Numeric Types: These data types represent numeric values and include integers, floats, and complex numbers.

- Integer (int): Whole numbers, such as 1, 2, 3, 4, 5.
- Float (float): Floating-point numbers, such as 3.14, 1.5, 0.01.
- Complex (complex): Numbers with a real and imaginary part, such as 2+3j.

Boolean Type: A Boolean value can only be True or False.

Sequence Types: These data types represent sequences of values and include strings, lists, and tuples.

- String (str): A sequence of characters, such as "Hello, World!".
- List (list): A mutable ordered sequence of values, such as [1, 2, 3, 4].
- Tuple (tuple): An immutable ordered sequence of values, such as (1, 2, 3, 4).

Set Types: Sets are used to store multiple items in a single variable. Sets are unordered, and the values are unique.

- Set (set): An unordered collection of unique items, such as {2, 1, 3, 4}.

Mapping Types: These data types represent mappings of keys to values and include dictionaries.

- Dictionary (dict): A collection of key-value pairs, such as {"name": "John", "age": 30}.

Primitive Data Types

In Python, primitive data types are simple data types that are built into the language and are not composed of other data types. They are also sometimes referred to as basic data types. Python has several primitive data types, including:

1. Numeric Types: These data types represent numeric values and include integers, floats, and complex numbers. These have been explained in detail in the previous answer.

2. Boolean Type: The Boolean data type represents a logical value that can be either True or False. It is often used in conditional statements, loops, and other logical operations.

In [1]:
```
x = True
y = False
```

```
In [2]: print(x)
        print(y)
```

```
True
False
```

3. String Type: The string data type represents a sequence of characters enclosed in quotation marks. Strings are used to represent text data in Python. Strings can be enclosed in single quotes ('...') or double quotes ("...").

```
In [4]: p = 'Hello, World!'
        q = "This is a string"
```

```
In [5]: print(p)
        print(q)
```

```
Hello, World!
This is a string
```

4. None Type: The None data type represents a null or empty value. It is often used to indicate that a variable has no value or has not been assigned a value.

```
In [6]: X = None
```

```
In [7]: print(X)
```

```
None
```

Type Conversion

Type conversion in Python, also known as typecasting, is the process of converting a value of one data type into another data type. Python supports both implicit and explicit type conversion.

1. Implicit Type Conversion:

Implicit type conversion, also known as coercion, is performed automatically by Python when necessary. For example, if we add an integer to a floating-point number, Python will convert the integer to a floating-point number before performing the addition.

```
In [1]: num_int = 10
        num_float = 2.5
        result = num_int + num_float
        print(result) # Output: 12.5
```

```
12.5
```

2. Explicit Type Conversion:

Explicit type conversion is the process of converting a value of one data type into another data type using a type conversion function. In Python, we can use the following built-in functions for explicit type conversion:

- int(): converts a value to an integer.
- float(): converts a value to a floating-point number.
- str(): converts a value to a string.
- bool(): converts a value to a Boolean value.

For example, we can convert an integer to a string using the str() function:

In [2]:
```
num_int = 10
num_str = str(num_int)
print(num_str) # Output: '10'
```

```
10
```

In [3]:
```
type(num_str)
```

Out[3]:
```
str
```

We can also convert a string to an integer using the int() function:

In [4]:
```
num_str = '10'
num_int = int(num_str)
print(num_int) # Output: 10
```

```
10
```

In [5]:
```
type(num_int)
```

Out[5]:
```
int
```

It's important to note that some type conversions may result in data loss or errors.

For example, converting a floating-point number to an integer using the int() function will truncate the decimal portion of the number, potentially resulting in data loss.

Additionally, converting a string that cannot be interpreted as a number to an integer or floating-point number will result in a ValueError exception.

In [6]:
```
name = len(input("Your Name? \t"))

print("Your name has " + str(name) + " characters")
```

```
Your Name?      srikanth
Your name has 8 characters
```

Exercise: Write a program that adds the digits in a 2 digit number

Example: If input is 35, then output should be 3 + 5 = 8

In [1]:
```
two_digit_number = input("Type your 2 digit number to add\t")

first_digit = two_digit_number[0]
second_digit = two_digit_number[1]

result = int(first_digit) + int(second_digit)
```

```
print(result)
```

```
Type your 2 digit number to add 28
10
```

Mathematical Operations Order

In Python, mathematical operations are performed using the standard order of operations, also known as PEMDAS (Parentheses, Exponents, Multiplication and Division, Addition and Subtraction).

The order of operations determines the sequence in which mathematical operations are executed. The order of operations is as follows:

1. Parentheses: expressions within parentheses are evaluated first.
2. Exponents: exponentiation is performed next.
3. Multiplication and Division: multiplication and division are performed from left to right.
4. Addition and Subtraction: addition and subtraction are performed from left to right.

For example, consider the following expression:

```
In [9]: result = 10 + 5 * 2 ** 3 / 4
```

Here, the order of operations is as follows:

1. First, the expression inside the parentheses is evaluated:

```
In [11]: 2 ** 3

Out[11]: 8
```

2. Next, multiplication and division are performed from left to right:

```
In [12]: 5 * 8

Out[12]: 40

In [13]: 40/4

Out[13]: 10.0
```

3. Finally, addition is performed:

```
In [14]: 10 + 10

Out[14]: 20
```

Therefore, the final value of result is 20.

It's important to note that the order of operations can be changed using parentheses. Expressions inside parentheses are always evaluated first, regardless of the order of operations. For example:

In [15]:
```
result = (10 + 5) * 2  # evaluates to 30
```

In this example, the addition inside the parentheses is performed first, resulting in 15. The multiplication is then performed, resulting in 30.

BMI Calculator Exercise

Write a program that calculates the Body Mass Index (BMI) from a user's weight and height.

The BMI is a measure of someone's weight taking into account their height. e.g. If a tall person and a short person both weigh the same amount, the short person is usually more overweight.

The BMI is calculated by dividing a person's weight (in kg) by the square of their height (in m):

$$BMI = \frac{weight\,(kg)}{height^2\,(m^2)}$$

In [16]:
```
Height = input("Enter your height: \t")
Weight = input("ENter your weight: \t")

bmi = (int(Weight)/float(Height)**2)

bmi_as_int = int(bmi)

print("Your BMI is ", str(bmi_as_int))
```

```
Enter your height: 	5.9
ENter your weight: 	85
Your BMI is  2
```

Number Manipulation

In [17]:
```
print(int(8/3))
```

```
2
```

In [18]:
```
print(round(8/3))
```

```
3
```

In [23]:
```python
print(8//3)
```

```
2
```

In [24]:
```python
score = 0
score = score + 1
# score += 1

print("Your score is " + str(score))
```

```
Your score is 1
```

F-string

f-strings (or formatted string literals) are a feature introduced in Python 3.6 that provide a convenient way to embed expressions inside string literals. F-strings are created by prefixing a string with the letter "f" or "F" and placing expressions inside curly braces {}.

F-strings allow for easy string interpolation by allowing the embedding of variables and expressions directly inside a string. They are also very readable and reduce the amount of boilerplate code needed for string formatting.

Here is an example of using an F-string to embed a variable:

In [25]:
```python
score = 0
height = 1.8
isWinning = True
```

In [27]:
```python
print(f"your score is {score}, your height is {height}, and you are winning is {
```

```
your score is 0, your height is 1.8, and you are winning is True
```

Exercise - Life in Weeks

Create a program using maths and f-Strings that tells us how many days, weeks, months we have left if we live until 90 years old.

It will take your current age as the input and output a message with our time left in this format:

You have x days, y weeks, and z months left.

Where x, y and z are replaced with the actual calculated numbers.

In [2]:
```python
lifespan = 90

age = input("What is your current age?\t")
remaining_years = int(lifespan) - int(age)
months = remaining_years * 12
weeks = remaining_years * 56
days = remaining_years * 365

print(f'You have {days} days, {weeks} weeks, and {months} months remaining')
```

```
What is your current age?       27
You have 22995 days, 3528 weeks, and 756 months remaining
```

Project - Tip Calculator

If the bill was $150.00, split between 5 people, with 12% tip.

Each person should pay (150.00 / 5) * 1.12 = 33.6 Format the result to 2 decimal places = 33.60

Tip: There are 2 ways to round a number. You might have to do some Googling to solve this.

In [33]:
```
print("Welcome to the Tip calculator")

bill = float(input("What was the total bill? \t"))

tip = int(input("How much you guys want to give the waiter? 10, 12,15 ?\t"))

people = int(input("How many people would you like to share the bill?\t"))

tip_as_percent = tip/100

total_tip = bill * tip_as_percent

total_bill = bill + total_tip

bill_per_person = total_bill/people

final_amount = "{:.2f}".format(bill_per_person)

print(f'Each person should pay: ${final_amount}')
```

```
Welcome to the Tip calculator
What was the total bill?        150
How much you guys want to give the waiter? 10, 12,15 ?  12
How many people would you like to share the bill?       5
Each person should pay: $33.60
```