**Assignment3**

**Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets**

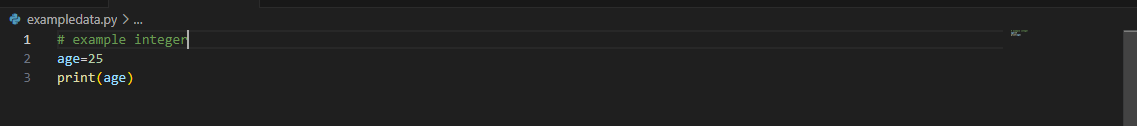
**Define data types:**

In Python, data types are used to categorize and classify data items based on their structure and characteristics. This helps computer systems process information more easily. Data types also determine the type of values that variables can store and the operations that can be performed on those values.

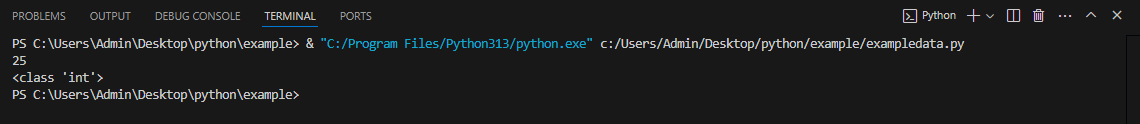
**Integers:**

Integers are whole numbers without a fractional part. They can be positive, negative, or zero.

Example:



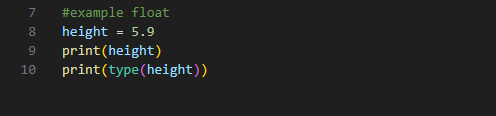
Output:



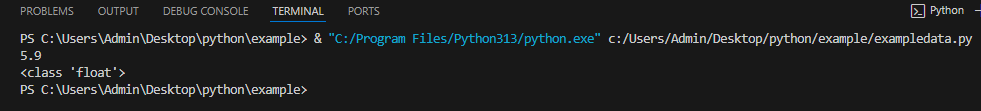
**Float:**

In Python, a **float** is a numerical data type that represents a **floating-point number**. A floating-point number is a number that has a decimal point or is in exponent notation, indicating digits before and after the decimal point or exponent. For example, 1.23 is a floating-point number with one digit before the decimal point and two digits after it

Example:



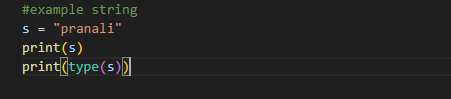
Output:



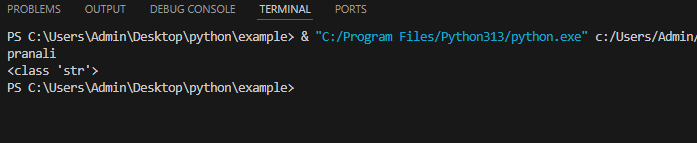
Strings:

A string is a sequence of characters. Python treats anything inside quotes as a string. This includes letters, numbers, and symbols. Python has no character data type so single character is a string of length 1.

Example:



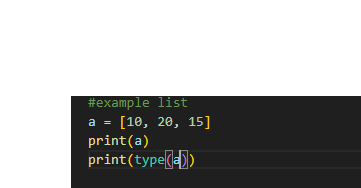
Output:



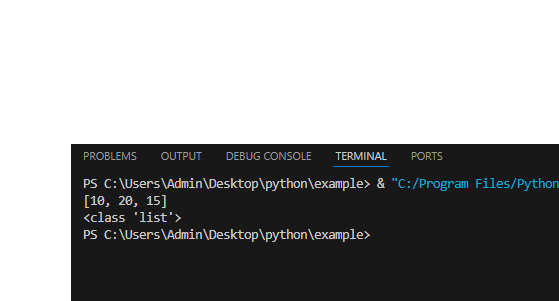
List:

In Python, a**list**is a built-in dynamic sized array (automatically grows and shrinks) that is used to store an ordered collection of items. We can store all types of items (including another list) in a list. A list may contain mixed type of items, this is possible because a list mainly stores references at contiguous locations and actual items maybe stored at different locations.

Example:



Output:



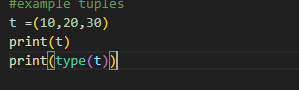
Tuples:

Python Tuple is a collection of objects separated by commas. A tuple is similar to a Python list in terms of indexing, nested objects, and repetition but the main difference between both is Python tuple is immutable, unlike the Python list which is mutable.

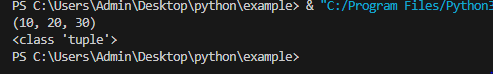
Characteristics of tuple:

* We cannot update items to a tuple once it is created.
* Tuples cannot be appended or extended.
* We cannot remove items from a tuple once it is created.

Example:



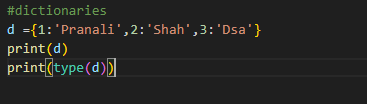
Output:



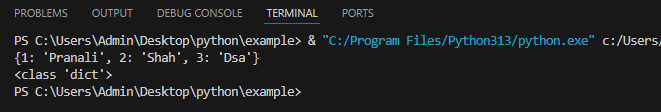
Dictionaries

**A Python dictionary**is a data structure that stores the value in **key: value** pairs. Values in a dictionary can be of any data type and can be duplicated, whereas keys can’t be repeated and must be immutable.

Example:



Output:



Sets:

Sets in Python are collections of unordered and unique elements.

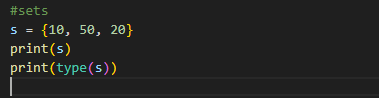
They are mutable, allowing for the addition and removal of items.

Sets are optimized for checking if an element is part of the set.

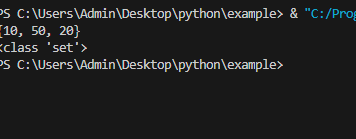
They can contain heterogeneous elements and do not allow duplicates.

Useful for mathematical set operations like union, intersection, and difference.

Example:



Output:



• Python variables and memory allocation.

Python variables:

In [Python](https://www.geeksforgeeks.org/python-programming-language-tutorial/), variables are used to store data that can be referenced and manipulated during program execution. A variable is essentially a name that is assigned to a value. Unlike many other programming languages, Python variables do not require explicit declaration of type. The type of the variable is inferred based on the value assigned.

Example for variable

X=5

print(x)

#it will create store 5 into x

Variables act as placeholders for data. They allow us to store and reuse values in our program.

* Variable names can only contain letters, digits and underscores (\_).
* A variable name cannot start with a digit.
* Variable names are case-sensitive (myVar and myvar are different).
* Avoid using [Python keywords](https://www.geeksforgeeks.org/python-keywords/) (e.g., if, else, for) as variable names

Assigning value to variable

**Dynamic Typing**

Python variables are dynamically typed, meaning the same variable can hold different types of values during execution.

Example:

x = 10

x = "Now a string".

### Assigning the Same Value

Python allows assigning the same value to multiple variables in a single line, which can be useful for initializing variables with the same value.

Example:

a = b = c = 100

print(a, b, c)

### Assigning Different Values

We can assign different values to multiple variables simultaneously, making the code concise and easier to read.

Example:

x, y, z = 1, 2.5, "Python"

print(x, y, z)

Output:

1 2.5 Python

Python operators

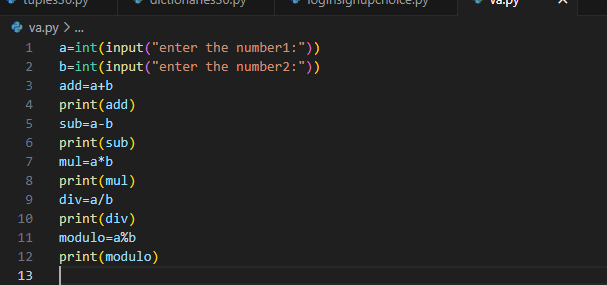
1 Assignment operator:

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.

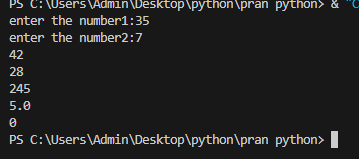
List of operator:

| **Operator** | **Description** | **Syntax** |
| --- | --- | --- |
| + | Addition: adds two operands | x + y |
| – | Subtraction: subtracts two operands | x – y |
| \* | Multiplication: multiplies two operands | x \* y |
| / | Division (float): divides the first operand by the second | x / y |
| // | Division (floor): divides the first operand by the second | x // y |
| % | Modulus: returns the remainder when the first operand is divided by the second | x % y |
| \*\* | Power: Returns first raised to power second | x \*\* y |

Example:



Output:

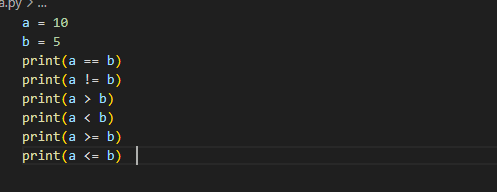


### 2) Comparison Operators:

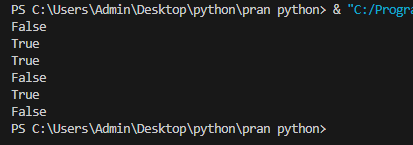
These operators are used to compare two values.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| == | Equal to | a == b |
| != | Not equal to | a != b |
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater than or equal to | a >= b |
| <= | Less than or equal to | a <= b |

Example:



Output:



**3. Logical Operators:**

These operators are used to combine conditional statements.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| and | Logical AND | a > 5 and b < 10 |
| or | Logical OR | a > 5 or b < 10 |
| not | Logical NOT | not(a > b) |

Example:

a = 10

b = 5

print(a > 5 and b < 10) # Output: True

print(a > 5 or b > 10) # Output: True

print(not(a > b)) # Output: False

### 4. Bitwise Operators:

These operators are used to perform bit-level operations on integers.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| & | Bitwise AND | a & b |
| ` | ` | Bitwise OR |
| ^ | Bitwise XOR | a ^ b |
| ~ | Bitwise NOT | ~a |
| << | Left shift | a << b |
| >> | Right shift | a >> b |

**Example:**

a = 10 # 1010 in binary

b = 4 # 0100 in binary

print(a & b) # Output: 0 (1000 & 0100 = 0000)

print(a | b) # Output: 14 (1010 | 0100 = 1110)

print(a ^ b) # Output: 14 (1010 ^ 0100 = 1110)

print(~a) # Output: -11 (~1010 = -1011 in two's complement)

print(a << 1) # Output: 20 (1010 << 1 = 10100)

print(a >> 1) # Output: 5 (1010 >> 1 = 0101)