HOW TO DOWNLOAD PYTHON-

Go to this link- https://www.python.org/downloads/

Download latest version of python and install it

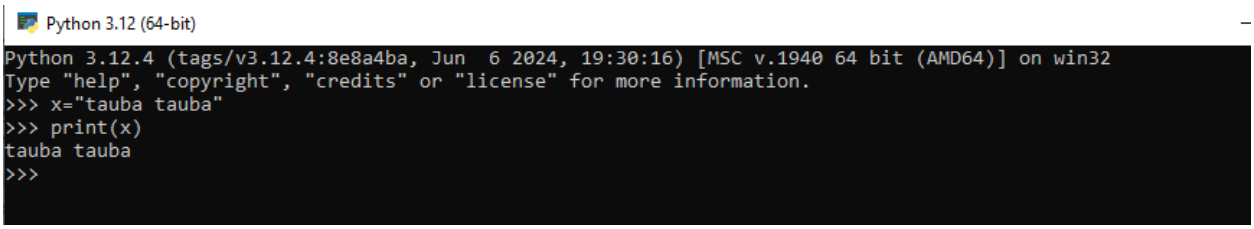
Day 1-

Constants- Fixed values such as Numbers, letters and strings

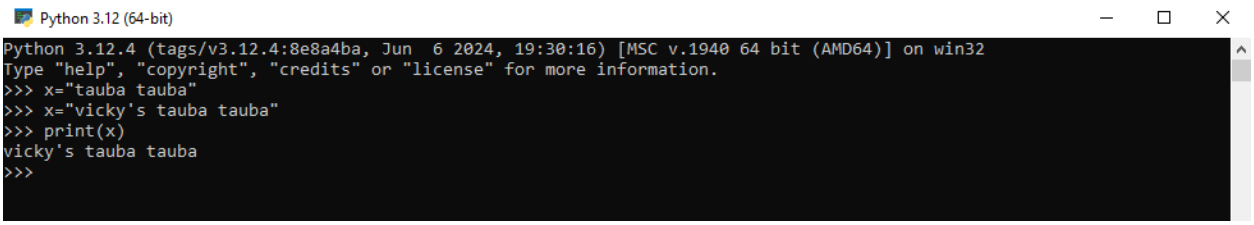
You can use single quotes(‘) or double quotes (“) for string constants

Variable- variable basically oka space where you can put your own value and call that whenever you want. It can be changed whenever you want

Assigning value to a variable-

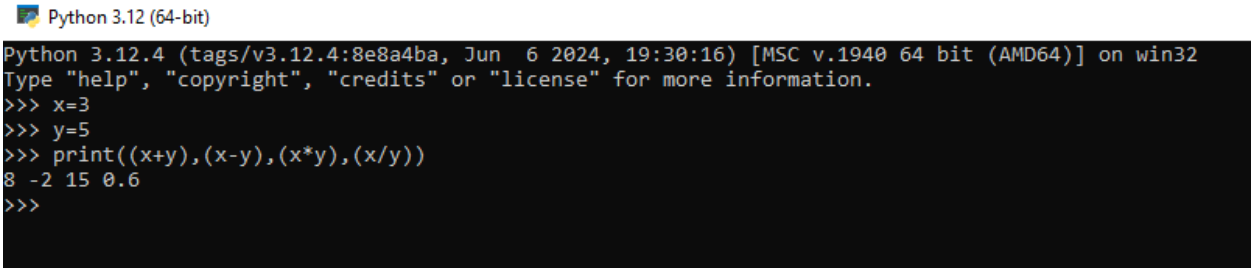


We assigned tauba tauba to x



You can also add two strings

You can also do addition, subtraction, multiplication and division of numbers-



Python is Case sensitive-

```
>>> x=5
>>> X=2
>>> print(x+X)
7
>>>
```

Successfully completed day 1! GOOD JOB

DAY2 PYTHON

Integers: Counting apples in a basket.

Floats: Measuring ingredients in a recipe (e.g., 1.5 cups of flour).

Strings: Writing a name or a sentence.

Booleans: Answering yes/no questions (e.g., Is the light on? True/False).

```
>>> x=1
>>> y=1.5
>>> z="chocolate"
>>> is_sweet= True
```

X is int

Y is float

Z is string

Is_sweet is Boolean

Conversion- INT to FLOAT-

```
>>> print(float(x))
1.0
```

Conversion- INT to STRING-

```
>>> print(str(x))
1
```

Conversion- STRING to Boolean

```
>>> print(bool(z))
True
>>>
```

True because z has “chocolate” saved if z was empty (z=””) then the value would return false

Conversion- Float to INT

```
>>> print(int(y))
1
>>>
```

Int datatype considers the value 1.5 as whole number 1

Congratulations GUYS DAY 2 completed.

DAY3 PYTHON

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators

Arithmetic operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

```
>>> a = 10
>>> b = 3
>>>
>>> print("Addition:", a + b)
Addition: 13
>>> print("Subtraction:", a - b)
Subtraction: 7
>>> print("Multiplication:", a * b)
Multiplication: 30
>>> print("Division:", a / b)
Division: 3.3333333333333335
>>> print("Modulus:", a % b)
Modulus: 1
>>> print("Exponentiation:", a ** b)
Exponentiation: 1000
>>> print("Floor Division:", a // b)
Floor Division: 3
>>>
```

Assignment operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3 print(x)

Comparison operators-

```
>>> a = 10
>>> b = 3
>>>
>>> print("Equal to:", a == b)
Equal to: False
>>> print("Not equal to:", a != b)
Not equal to: True
>>> print("Greater than:", a > b)
Greater than: True
>>> print("Less than:", a < b)
Less than: False
>>> print("Greater than or equal to:", a >= b)
Greater than or equal to: True
>>> print("Less than or equal to:", a <= b)
Less than or equal to: False
>>>
```

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

```
>>> biryani = 25
>>>
>>> if biryani < 30 and biryani > 20:
...     print("Let's eat biryani, it is " + str(biryani))
...
Let's eat biryani, it is 25
>>>
```

```
>>> biryani = 35
>>>
>>> if biryani < 20 or biryani > 30:
...     print("Biryani is not in the ideal price range, it is " + str(biryani))
...
Biryani is not in the ideal price range, it is 35
>>>
```

```
>>> biryani = 18
>>>
>>> if not (20 < biryani < 30):
...     print("Biryani is not in the ideal price range, it is " + str(biryani))
...
Biryani is not in the ideal price range, it is 18
>>>
```

Congratulations Crazy coders successful ga python day 3 ayipoindhi!

DAY 4 PYTHON-

Lists-

Ordered and Indexable:

Lists are ordered collections of items. Each item in a list has a specific position, called an index, starting from 0. You can access items using their index. For example, my_list[0] accesses the first item in the list.

Mutable:

Lists can be modified after their creation. You can change the value of an item, add new items, or remove items. For example, my_list.append("new_item") adds a new item to the end of the list, and my_list[1] = "changed_item" changes the value of the second item.

Heterogeneous:

Lists can contain items of different data types, including integers, strings, floats, and even other lists. This flexibility allows you to store diverse data within a single list. For example, my_list = [1, "hello", 3.14, [1, 2, 3]] contains an integer, a string, a float, and another list.

```
>>> my_list=["apples","bananas","oranges"]
>>> print(my_list)
['apples', 'bananas', 'oranges']
```

```
>>> my_list.append("onions")
>>> print(my_list)
['apples', 'bananas', 'oranges', 'onions']
>>> my_list.remove("oranges")
```

```
>>> my_list.remove("oranges")
>>> print(my_list)
['apples', 'bananas', 'onions']
>>>
```

```
>>> my_list[1]="oranges"
>>> print(my_list)
['apples', 'oranges', 'onions']
```

Tuples-

Tuples are immutable, meaning you cannot change, add, or remove items after the tuple is created.

Tuples can contain items of different data types.

Tuples can be nested, meaning you can have tuples within tuples.

```
>>> my_list=tuple(my_list)
>>> my_list[1]="bananas"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> print(my_list)
('apples', 'oranges', 'onions')
```

```
>>> my_tuple_grocery=("butter","eggs","milk")
>>> print(my_tuple_grocery)
('butter', 'eggs', 'milk')
```

Congratulations techies Day 4 kuda ayipoindhi

DAY5 PYTHON-

A function is like a recipe that tells your computer how to perform a specific task. You define the steps once, and then you can reuse them whenever you want.

```
def ingredients():
    print("garlic,bread,butter,chillies")
```

```
>>> def ingredients():
...     print("garlic,bread,butter,chillies")
...
>>> ingredients()
garlic,bread,butter,chillies
>>>
```

Functions can take inputs, called parameters, to perform tasks based on the given values. Here's an example:

```
>>> def greet(name):
...  print(f"Hello {name} welcome to cheezy techie's friend's party")
...
>>> greet("vikasa")
Hello vikasa welcome to cheezy techie's friend's party
>>>
```

Here we are calling a function and giving our own input.

Why Use Functions?

- **Reusability**: Write code once and use it multiple times.
- **Organization**: Break your code into smaller, manageable parts.
- **Readability**: Makes your code easier to read and understand