PHASE 1 PART A – Core Python (Beginner to Intermediate)

Presented by: JISHA VARGHESE (Data Science Enthusiast)

Lesson 1: Your First Python Code

```
print("hello, world")
```

```
hello, world
```

```
# Different ways - Print() function
print ('hello world')
# CASE 1 : print (' You 're a good man') - shows error as three quotes
# Solution
print ("You 're a good man")
# CASE 2 : print("You 're a "good" person") - Error as in between
"good" quotes,python again gets confused.
# Solution :
print ('''You 're a "good" person''')
```

```
hello world
You 're a good man
You 're a "good" person
```

Lesson 2: Variables and Data Types

A variable is a name used to store data. It's like a container in memory.

```
name = 'Juvana'      # String (Text)
age = 25             # integer (Number)
height = 5.3         # float (Decimal)
is_student = True    # Boolean (True or false)
print(name)
# print(Name) Python is case sensitive (Error found)
print("Juvana age is",age)
```

```
Juvana
Juvana age is 25
```

```
a,b,c=20,"Jisha",33  # assign multiple variables
print(a,b,c)
```

```
20 Jisha 33
```

Rules for Naming Variables in Python (Very Important):

You can:

1.Use letters (A–Z, a–z)

2.Use numbers (0–9) — but not at the start

3.Use underscore _ (recommended for separating words)

You cannot:

1.Start a variable with a number

2.Use special characters like @, #, %, !

3.Use Python keywords like if, for, print, while, etc.

```
# Ways to declare a variable name
MyName = "JISHA" #Pascal Case
myName = "JISHA" # Camel Case
myname = "JISHA" # Flat Case
my_name = "Jisha" # Snake case
```

LESSON 3: print() and input() Functions + Data Types

What is a function

What is print() and why we use it

What is input() and how to take user input

Different data types in Python

How to check and convert data types

1. What is a Function?

A function is a block of code that performs a specific task.

You can call a function to run it.

There are two types:

Built-in functions: like print(), input(), len()

User-defined functions: we'll learn later

Calling a function means writing its name followed by ()

2. print() – Built-in Function

Purpose:

Used to display output on the screen.

```
print("Hello, World")
Hello, World
print(5+3)
8
```

3. input() – Built-in Function

Purpose:

Used to take input from the user as a string (text).

name = input("Enter your name: ") print("Hello,", name)

Even if you enter a number, input() will return it as a string.

```
name=input("Enter your name:")
```

```
Enter your name: Jisha Varghese
```

```
print("Hello" ,name, "nice to meet you", sep="-")
```

```
Hello-Jisha Varghese-nice to meet you
```

```
Customer_ID=input("Enter your ID")
```

```
Enter your ID 123D34RR
```

```
print(name,Customer_ID,"Successfully created")
```

```
JISHA VARGHESE 123D34RR Successfully created
```

4. Data Types in Python - using type() function

```
| **Type** | **Example** | **Use**          |
| -------- | ----------- | ---------------- |
| `int`    | 5, 100      | Whole numbers    |
| `float`  | 5.5, 3.14   | Decimal numbers  |
| `str`    | "Hello"     | Text/words       |
| `bool`   | True, False | Yes/No logic     |
```

5. Type Checking and Conversion

```
X=5
type(X)
```

```
int
```

```
print(type(name))
```

```
<class 'str'>
```

```
print(name , "Customer ID:", Customer_ID,"is", True)
```

```
JISHA VARGHESE Customer ID: 123D34RR is True
```

```
Checking = print(name , "Customer ID:", Customer_ID,"is", True)
```

```
JISHA VARGHESE Customer ID: 123D34RR is True
```

```
print(type(Checking)) # Note : You are assigning the result of
print(), not the text it shows – and print() returns None by default.
```

```
<class 'NoneType'>
```

```
Checking = name , "Customer ID:", Customer_ID,"is", True
```

```
print(Checking)
```

```
('JISHA VARGHESE', 'Customer ID:', '123D34RR', 'is', True)
```

```
print(type(Checking))
```

```
<class 'tuple'>
```

```
Checking = name +  "Customer ID:" + Customer_ID + "is True"
```

```
print(type(Checking))
```

```
<class 'str'>
```

Type Conversion:

```
age =input("Enter your age:")
```

```
Enter your age: 22
```

```
print(type(age))   # input is string
```

```
<class 'str'>
```

```
# Convert string to int
age =int(age)
```

```
print(type(age))
```

```
<class 'int'>
```

```
# Convert to float
pi = input("Enter value:")
```

```
Enter value: 3.14
```

```
print(type(pi))
```

```
<class 'str'>
```

```
pi =float(pi) # can also write pi = float("3.14")
```

```
print(type(pi))
```

```
<class 'float'>
```

```
pi_1=int(pi)
```

```
print(type(pi_1))
```

```
<class 'int'>
```

```
print(type(pi))
```

```
<class 'float'>
```

TASK

```
# Take user input
name =input("Enter your name: ")
age = input("Enter your age:")

Enter your name:  Juvana
Enter your age: 5

# Convert age to int
print(type(age))
age =int(age)
print(type(age))

<class 'str'>
<class 'int'>

# Print info
print("Hello", name + "!")
print("You are", age, "years old.")
print("Data type of age is:", type(age))

Hello Juvana!
You are 5 years old.
Data type of age is: <class 'int'>
```

Lesson 4: Operators and Expressions in Python

What Are Operators?

Operators are symbols or words that perform operations on variables or values.

Just like in math:

5 + 3 → here `+` is an operator

| Operator Type | Description | Example | |

| ---------- | ---------- | ---------- | ---------- |

| **1. Arithmetic** | Basic math: +, -, *, /, % | `5 + 3` | |

| **2. Assignment** | Assign values to variables | `x = 10` | |

| **3. Comparison** | Compare values: ==, !=, >, < | `a > b` | |

| **4. Logical** | Combine conditions: and, or, not | `x > 5 and y < 10` | |

| **5. Identity** | Check memory location: `is`, `is not` | `a is b` | |

| **6. Membership** | Check if value in sequence: `in`, `not in` | `"a" in "apple"` | |

| **7. Bitwise** | Operate on binary digits (advanced) | `&`, `|`, `^` |

1. Arithmetic Operators

```
| Operator | Meaning        | Example  | Output |
| -------- | -------------- | -------- | ------ |
| `+`      | Addition       | `5 + 3`  | `8`    |
| `-`      | Subtraction    | `5 - 3`  | `2`    |
| `*`      | Multiplication | `5 * 3`  | `15`   |
| `/`      | Division       | `5 / 2`  | `2.5`  |
| `//`     | Floor Division | `5 // 2` | `2`    |
| `%`      | Modulus        | `5 % 2`  | `1`    |
| `**`     | Power          | `2 ** 3` | `8`    |
```

2. Assignment Operators

```
| Operator | Example  | Meaning              |
| -------- | -------- | -------------------- |
| `=`      | `x = 5`  | Assign 5 to x        |
| `+=`     | `x += 2` | Same as `x = x + 2`  |
| `-=`     | `x -= 2` | Same as `x = x - 2`  |
| `*=`     | `x *= 2` | Same as `x = x * 2`  |
| `/=`     | `x /= 2` | Same as `x = x / 2`  |
```

3. Comparison Operators

```
| Operator | Example  | Meaning              | Result  |
| -------- | -------- | -------------------- | ------- |
| `==`     | `5 == 5` | Equal                | `True`  |
```

`!=`	`5 != 3`	Not equal	`True`
`>`	`5 > 3`	Greater than	`True`
`<`	`5 < 3`	Less than	`False`
`>=`	`5 >= 5`	Greater than or equal	`True`
`<=`	`3 <= 5`	Less than or equal	`True`

4. Logical Operators

Operator	Description	Example
`and`	True if both are true	`x > 3 and y < 10`
`or`	True if at least one is true	`x > 3 or y < 2`
`not`	Reverses the result	`not(x > 3)`

5. Membership Operators

Operator	Meaning	Example	Output
`in`	Value exists in sequence	`'a' in 'apple'`	True
`not in`	Value not in sequence	`'z' not in 'apple'`	True

6. Identity Operators (Used less often)

Operator	Description	Example
`is`	True if both are same object	`a is b`
`is not`	True if not same object	`a is not b`

Practice Task 1:

```
a = 10
b = 5
c = "Python"

# 1. Add a and b
a = 10
b =  5
x = a+b
print("Total=", x)

Total= 15

# 2. Check if a is greater than b
a = 10
b = 5
```

```
x = a>b
print ( "Is", a , "greater than ", b, "?", "Answer:", x)
```

```
Is 10 greater than  5 ? Answer: True
```

```
# 3. Check if "t" is in c
c = "Python"
print("Is t is in c?" , "t" in c)
```

```
Is t is in c? True
```

```
# 4. Use floor division on a and b
a = 10
b = 3
x = a//b
print (x)
```

```
3
```

```
# 5. Check if a is not equal to b
a = 10
b = 5
f = a != b
print(f)
# option 2 :
print(not(a == b))
```

```
True
True
```

```
# 6. What will this print?
name = "python"
print("o" in name)
```

```
True
```

Practice Task 2:

```
x = 8
y = 3
print(x % y)
print(x ** y)
print(x > 5 and y < 5)
print(not(x < 5))
```

```
2
512
True
True
```

Lesson 5: Data Types Deep Dive in Python

```
Python is dynamically typed, meaning you don't have to declare the
type of a variable — Python figures it out automatically.
But understanding data types is crucial because they determine:
What kind of data can be stored
What operations can be performed
```

Why Are Data Types Important in Python (and Data Science)?

In Data Science, you'll work with:

Numbers (math, stats)

Text (customer names, categories)

Booleans (True/False logic)

Collections (lists, tables, etc.)

To analyze, clean, and model data, you must understand how each type behaves.

Python's Core Data Types

Category	Data Type	Example
Numeric	`int`	`a = 10`
	`float`	`pi = 3.14`
	`complex`	`z = 2 + 3j`
Text	`str`	`name = "Jisha"`
Boolean	`bool`	`is_ready = True`
Sequence	`list`	`marks = [10, 20, 30]`
	`tuple`	`coords = (3, 4)`
Set	`set`	`nums = {1, 2, 3}`
Mapping	`dict`	`data = {"name":"Ana"}`
NoneType	`NoneType`	`val = None`

1. Numeric Types

int: Whole numbers → a = 10

float: Decimal → pi = 3.14

complex: Imaginary part → z = 2 + 3j

```
a1 = int(10)
print(a1)
```

```
print(type(a1))
 # or can also be written
a2 = 20
print(a2)
type(a2)
```

```
10
<class 'int'>
20
```

```
int
```

```
b2 = 3.14
print(b2)
type(b2)
b3 = float(3)
print(b3)
print(type(b3))
```

```
3.14
3.0
<class 'float'>
```

```
c1 = (3,4)
print(c1)
print(type(c1))
c2 = complex(3,4)
print(c2)
print(type(c2))
```

```
(3, 4)
<class 'tuple'>
(3+4j)
<class 'complex'>
```

2. String (str)

```
name = "Jisha"
print(name.upper())
```

```
JISHA
```

3. Boolean (bool)

```
x = True
y = False
# Used in conditions, filtering, machine learning labels (yes/no).
```

```
bool1 = True
bool2 = False
print(bool1)
```

```
print(type(bool1))
print(bool2)
print(type(bool2))
```

```
True
<class 'bool'>
False
<class 'bool'>
```

4. List

```
fruits = ["apple", "banana", "cherry"]
print(fruits)
print(fruits[0])
print(fruits[1])
print(fruits[0], fruits[1])
```

```
['apple', 'banana', 'cherry']
apple
banana
apple banana
```

Difference between tuple , list & Set

```
c3 = [1,2,3,'jisha']
print(c3)
print(type(c3))
```

```
[1, 2, 3, 'jisha']
<class 'list'>
```

```
c4 = (1,2,3,'jisha') # here two data types are here in () which is considered as tuple
print(c4)
print(type(c4))
```

```
(1, 2, 3, 'jisha')
<class 'tuple'>
```

```
c5= {1,2,3,'jisha'}
print(c5)
print(type(c5))
```

```
{1, 2, 3, 'jisha'}
<class 'set'>
```

Lists are mutable (can be changed), and used a lot in data analysis.

```
fruits = ["apple","banana","cherry"]
fruits[0]="orange"   # changing "banana" to "orange"
print(fruits)
```

```
['orange', 'banana', 'cherry']
```

```
You can also:
Add items
Remove items
Sort items
Reverse the list

Why is this useful in data analysis?
In data analysis, we often work with:
Tables (like rows of data)
Lists of numbers or strings
Results from filtering or computations
And we need to change them during processing.

ages =[25,30,28]
ages.append(34) # Add new age
ages[0] = 26 # update first age
print(ages)
```

```
[26, 30, 28, 34]
```

```
Opposite of Mutable? → Immutable
tuple is an example of an immutable type (once created, you can't
change it).
```

TASK

```
colors = ["red","green","blue","yellow"]

colors[1]="purple"
print(colors)
```

```
['red', 'purple', 'blue', 'yellow']
```

```
NOTE :If you know the value but not the index
You can find the index using .index(), then update the value.

numbers =[1,2,3,4,5,6,7]
index_of_5 = numbers.index(5)
numbers[index_of_5]= 99
print(numbers)
```

```
[1, 2, 3, 4, 99, 6, 7]
```

```
numbers.index(5) = 99  #  Invalid
 Why?
Because numbers.index(5) returns a value (an integer), but you can't
assign directly to a function call like that.
 What does this line mean:
numbers.index(5) = 99
You're saying:
```

“Find where 5 is in the list and put 99 there.”

Sounds logical, right? But Python doesn’t allow it directly. Why?

Let's break it down:
numbers.index(5)
This part only gives you the index number where 5 is found.
For example, if the list is:
numbers = [2, 5, 9]
Then:
numbers.index(5) # → returns 1
So, you're getting a value back — just the number 1.
But then you're trying to assign something to it like this:

1 = 99   # This is invalid!
And Python says:

"Hey! You can't assign a value to a number like 1. That doesn't make sense."

```
number =[1,2,3,4,56,44]
number.index(56)
```

```
4
```

```
number[4]=20

print(number)
```

```
[1, 2, 3, 4, 20, 44]
```

```
number=[2,4,5,6,4]
number.index(4)
```

```
1
```

```
number[1]=2 #it changes the first come out if the numbers are same
print(number)
```

```
[2, 2, 5, 6, 4]
```

```
colors = ["red", "green", "blue", "yellow"]

colors.append("pink")

print(colors)
```

```
['red', 'green', 'blue', 'yellow', 'pink']
```

```
problem
colors = ["red", "green", "blue", "yellow"]
index_1=colors[1]
index_1.append("oranage")
```

```
print(colors)
Imagine:
A list is like a shopping bag → you can put more items in it (append).
A string is like a single apple → you can’t put another apple inside it.
If you want another apple, you need to put it in the bag, not inside the apple.

.index(value) → Finds the position of a value in the list.
.insert(position, value) → Puts a new value at a specific position.
They are two different tools — .index() searches for something inside the list, while .insert() places something in.

colors = ["red", "green", "blue", "yellow"]
colors.insert(1,"orange")
print(colors)

['red', 'orange', 'green', 'blue', 'yellow']

# Q.5 Remove "blue" from the list.
colors = ["red", "green", "blue", "yellow"]
colors.remove("blue")
print(colors)

['red', 'green', 'yellow']

# 6. Reverse the entire list.
colors = ["red", "green", "blue", "yellow"]
colors.reverse()
print(colors)

#  In Python:
# colors.reverse → just points to the method (does nothing)
# colors.reverse() → calls the method and actually reverses the list

['yellow', 'blue', 'green', 'red']

# Q 7. Print the final list.
print(colors[-1])

red


| Value  | Index |
| ------ | ----- |
| yellow | 0     |
| blue   | 1     |
| green  | 2     |
| red    | 3     |

Negative indexing works like this:
```

```
-1 means last item → index 3 here → "red"
-2 means second last → index 2 → "green"
-3 means third last → index 1 → "blue"
and so on…

['yellow', 'blue', 'green', 'red']
   -4        -3       -2      -1

new_colors = colors[::-1]
print(new_colors)
```

```
['red', 'green', 'blue', 'yellow']
```

```
help(colors.append)
```

```
Help on built-in function append:

append(object, /) method of builtins.list instance
    Append object to the end of the list.
```

TIPS

Action	How	Example
Show methods	Type `.` + **Tab**	`colors.` → Tab
Autocomplete	Start typing + **Tab**	`colors.ap` → Tab
Method help	Inside `()` + **Shift+Tab**	`colors.append(`
Full help	`help()`	`help(colors.append)`
Run cell	**Shift+Enter**	–
New cell above	**A**	–
New cell below	**B**	–

| Action | What to Do | Example | Result |

| ------------------------------ | -- | ---------------------------- | -- |

| **See all available methods** | Type the variable + `.` and press **Tab** | `colors.` → Tab | Shows list of methods like `append`, `remove`, `reverse` |

| **Autocomplete a method** | Start typing, then press **Tab** | `colors.ap` → Tab | Completes to `colors.append` |

| **See method explanation** | Place cursor inside `()` and press **Shift + Tab** | `colors.append(` → Shift + Tab | Shows what the method does and parameters |

| **Get full method documentation** | Use `help()` | `help(colors.append)` | Prints detailed docstring in output |

| **Run a cell** | Press **Shift + Enter** | *(any code)* → Shift + Enter | Executes the code in the current cell |

| **Insert a new cell above** | Press **A** (in command mode) | N/A | Adds an empty cell above |

| **Insert a new cell below** | Press **B** (in command mode) | N/A | Adds an empty cell below |

5. Tuple

What is a Tuple?

A tuple is like a list, but immutable (cannot be changed after creation).

Written with round brackets ().

Can store different data types.

```
colors =("red","green","blue")
print(colors)
```

```
('red', 'green', 'blue')
```

```
my_tuple = (1, "apple", 3.5)
print(my_tuple)
```

```
(1, 'apple', 3.5)
```

Key Differences from Lists

Feature	List	Tuple
Brackets	`[]`	`()`
Mutable?	Yes	No
Faster?	No	Yes
Methods	Many	Few (`count`, `index`)

Creating Tuples

```
t1 =(1,2,3) # normal
t2 = ("a",) # Single element (Comma is a must)
t3 = tuple([4,5,5]) # from list
```

Accessing Tuple Elements

```
t = (1,5,6,4)
print(t[0])
print(t[-4])
```

```
1
1
```

Tuple Methods

```
t = (1,23,22,4,4)
print(t.index(22))
print(t.count(4))
print(t.a
```

```
2
2
```

```
# Q 1. Create a tuple with fruits: "apple", "banana", "cherry".
fruits = ("apple", "banana", "cherry")
print(fruits)
```

```
('apple', 'banana', 'cherry')
```

```
# Q 2. Print the 2nd element
fruits = ("apple", "banana", "cherry")
print(fruits[1])
```

```
banana
```

```
# Q3.Find the index of "banana".
print(fruits.index("banana"))
```

```
1
```

```
# Q 4. Count how many times "apple" appears.
print(fruits.count("apple"))
```

```
1
```

```
# 5. Loop and print all elements
for fruit in fruits:
    print(fruit)
```

```
apple
banana
cherry
```

Try changing "apple" to "yellow" — what happens?

colors[1] = "yellow" # Error: Tuples are immutable

This shows tuples cannot be modified.

```
# Q6. Check if "apple" exists in the tuple.
print("apple" in fruits)
```

```
True
```

```
# Q5. Find the length of the tuple.
print(len(fruits))
```

```
3
```

6. Set (Combination of elements and objects)

```
unique_nums = {1, 2, 3, 3}
print(unique_nums)
```

```
{1, 2, 3}
```

7. Dictionary (dict) - (KEY, VALUE)

```
student = {"name": "Ana", "age": 22}
print(student["name"])  # Output: Ana
```

```
Ana
```

```
print(type(3.14))          # float
print(type("hello"))       # str
print(type([1, 2, 3]))      # list
```

```
<class 'float'>
<class 'str'>
<class 'list'>
```

```
a = 7
b = 3.14
c = "Python"
d = [1, 2, 3]
e = {"name": "Jisha", "age": 25}
f = (10, 20)
g = {1, 2, 2, 3}
h = None

print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
print(type(f))
print(type(g))
print(type(h))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'list'>
<class 'dict'>
<class 'tuple'>
<class 'set'>
<class 'NoneType'>
```

TASK

Q 1. Write a python programming code that prints the following text exactly as it appears.

Python is fun.

"Quotes" and 'single quotes' can be tricky.

```
print('Python is fun.')
```

```
Python is fun.
```

```
# Option 1
print(''' "Quotes" and 'single quotes' can be tricky.''')
# Option 2
print(""" "Quotes" and 'single quotes' can be tricky.""")
```

```python
# Option 3
print("\"Quotes\" and 'single quotes' can be tricky.")
```

```
 "Quotes" and 'single quotes' can be tricky.
 "Quotes" and 'single quotes' can be tricky.
"Quotes" and 'single quotes' can be tricky.
```

```python
# If you want to combine
print("Python is fun.\"Quotes\" and 'single quotes' can be tricky.")
```

```
Python is fun."Quotes" and 'single quotes' can be tricky.
```

```python
# next line (Break line)
print("Python is fun.\n\"Quotes\" and 'single quotes' can be tricky.")
```

```
Python is fun.
"Quotes" and 'single quotes' can be tricky.
```

Q2. For a business create 3 variables to store-name, age and city.Then print a sentence that uses these variables.

```python
name = "Max"
age = 22
city = "Al ain"
print( "My name is" ,name, "and i'm" ,age, "lives in",city )
```

```
My name is Max and i'm 22 lives in Al ain
```

```python
# if  Dict used
person = {'name': 'Max' , "age":22, "city": "Al AIN"}

# Option 1: Using keys directly
print("my name is ",person["name"], "and i'm",person["age"],"lives in
",person["city"])
```

```
my name is  Max and i'm 22 lives in  Al AIN
```

```python
# Option 2: Using f-strings (formatted string)
# Here {} means → insert variable value here.Here no commas needed
just insert variable in {} .
print(f"my name is {name} and i'm {age} lives in {city}")
```

```
my name is Max and i'm 22 lives in Al ain
```

```python
# Adding expression
print(f"my name is {name} and i'm {age + 1} lives in {city}")
```

```
my name is Max and i'm 23 lives in Al ain
```

1. Escape Sequences (like \n, \t)

Escape sequences are special codes inside strings that do things other than showing normal text.

Escape Sequence	Meaning
`\n`	New line
`\t`	Tab space
`\\`	Backslash itself
`\'`	Single quote
`\"`	Double quote
`\b`	Backspace
`\r`	Carriage return (moves cursor to start of line)

```
print("Hello\nWorld") #\n → New line
```

```
Hello
World
```

```
print("Name:\tJisha") #\t → Tab space
```

```
Name: Jisha
```

```
print("C:\\Users\\Documents") #\\ → Backslash itself
```

```
C:\Users\Documents
```

```
print('It\'s a sunny day') #\' → Single quote
```

```
It's a sunny day
```

```
print("He said, \"Hello!\"") #\" → Double quote
```

```
He said, "Hello!"
```

```
print("Helloo\b") #\b → Backspace (removes previous character)
```

```
Hello
```

REVISION TASK

Lesson 1 – Basics

Q 1. Write code to print: Hello World.

```
A = "Hello World"
print(A)
```

```
Hello World
```

Q 2. Store your name in a variable and print it.

```
b = "Jisha Varghese"
print(b)
print(type(b))
```

```
Jisha Varghese
<class 'str'>
```

Lesson 2 – Operators

If a = 10, b = 3:

Find the remainder.

Check if a > b.

Use floor division.

```
a = 10
b = 3
print(a % b) # remainder → 1
print(a>b)
print(a//b)
```

```
1
True
3
```

Lesson 3 – Input & Output

Ask the user for their name and print:

"Hello , nice to meet you!" using an f-string.

```
name = input(" Enter your name:")
print(f"Hello {name} nice to meet you!")
```

```
 Enter your name: Jisha Varghese
Hello Jisha Varghese nice to meet you!
```

Lesson 4 – Lists

Start with:

colors = ["red", "green", "blue"]

Add "yellow".

Replace "green" with "purple".

Print the last element using negative index.

```
colors = ["red","green","blue"]
colors.append("yellow") # add

print(colors)

['red', 'green', 'blue', 'yellow']

colors[1]="purple" # Replace

print(colors)

['red', 'purple', 'blue', 'yellow']

print(colors[-1])
print(colors)

yellow
['red', 'purple', 'blue', 'yellow']
```

Lesson 5 – Tuples

Create a tuple of fruits: "apple", "banana", "cherry".

Print the 2nd element.

Count how many times "apple" appears.

```
fruits = ("apple","banana","cherry")
print(fruits[1])


banana

print(fruits.count("apple"))

1
```

Lesson 6 – Sets & Dictionaries

Make a set with {1, 2, 2, 3} and print it.

Create a dictionary:

student = {"name": "Ali", "age": 20}

Print only the "age".

```
set = {1,2,3}
print(set)
```

```
{1, 2, 3}
```

```
student = {"name":"Ali","age":20}
print(student["age"])
```

```
20
```

```
print(f"My name is {student["name"]} and i'm {student["age"]}")
```

```
My name is Ali and i'm 20
```

NOTES

```
# [] = access (indexing, slicing)

# () = functions / tuples

# {} = sets / dictionaries

# Square Brackets []

# Used for indexing / accessing elements in lists, tuples, dicts.

colors = ["red", "blue"]
print(colors[0])
student = {"name": "Ali"}
print(student["name"])
```

```
red
Ali
```

```
# Parentheses ()

# Used for functions/methods (append(), count(), index()) or creating
tuples.

colors.append("green")  # add new item to list
print(colors)
```

```python
fruits = ("apple", "banana") # tuple
print(fruits)
```

```
['red', 'blue', 'green', 'green', 'green']
('apple', 'banana')
```

```python
# Curly Braces {}

# Used for sets and dictionaries.

s = {1, 2, 3}  # set
student = {"name": "Ali", "age": 20}  # dict
print(s)
print(student)
```

```
{1, 2, 3}
{'name': 'Ali', 'age': 20}
```

Mini Real-Life Project: Student Database

Question / Task

You are asked to create a small Student Database in Python using lists, tuples, sets, and dictionaries together.

Your database should allow you to:

1. Store details of multiple students.

Each student should have:

Name (string)

Age (integer)

Subjects (tuple of subjects, since subjects are usually fixed for that student).

2. Add a new student to the list.

3. Print the name and age of each student.

4. Create a set of all unique subjects across all students (no duplicates).

5. Find and print the age of a specific student (e.g., "Sara").

```
# Store students as a list of dictionaries
students = [
    {"name":"Ali","Age":22,"Subject":("English","Chemistry")},
    {"name":"Jisha","Age": 23,"Subject":("Science","Maths")}
]

print(students)
```

```
[{'name': 'Ali', 'Age': 22, 'Subject': ('English', 'Chemistry')},
{'name': 'Jisha', 'Age': 23, 'Subject': ('Science', 'Maths')}]
```

```
# Step 2: Add a new student (append to list)
students.append({"name":"Sara","Age":20,"Subject":
("English","Science")})
print(students)
```

```
[{'name': 'Ali', 'Age': 22, 'Subject': ('English', 'Chemistry')},
{'name': 'Jisha', 'Age': 23, 'Subject': ('Science', 'Maths')},
```

```
{'name': 'Jisha', 'Age': 20, 'Subject': ('English', 'Science')},
{'name': 'Jisha', 'Age': 20, 'Subject': ('English', 'Science')},
{'name': 'Sara', 'Age': 20, 'Subject': ('English', 'Science')}]
```

```
# Step 3: Print all student names (looping)
for student in students:
    print(f"Name: {student['name']} | Age: {student['Age']}")
```

```
Name: Ali | Age: 22
Name: Jisha | Age: 23
Name: Jisha | Age: 20
Name: Jisha | Age: 20
```

```
# Step 4: Make a set of all unique subjects

# NOTE :
# A set in Python:
# Stores only unique values → duplicates are automatically removed.
# The order is not guaranteed (it may shuffle).

all_subjects = set()
for student in students:
    all_subjects.update(student["Subject"])

print("Unique subjects:", all_subjects)
```

```
Unique subjects: {'Science', 'Maths', 'Chemistry', 'English'}
```

```
# Step 5: Find the age of "Sara"
for student in students:
    if student["name"] == "Sara":
        print("Sara's age is:", student["Age"])
```

```
Sara's age is: 20
```