Python

Python is a popular programming language.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

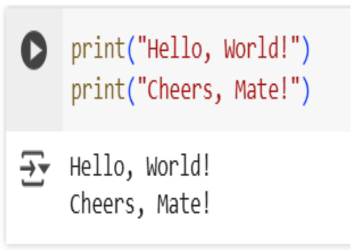
- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.

Why use Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

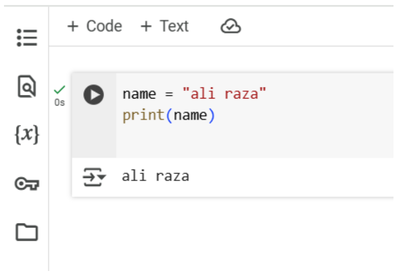
Coding in Python using Google Colab

The Python **print()** function is often used to output variables.



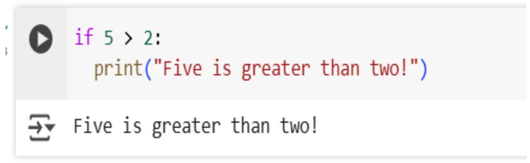
```
print("Hello, World!")
print("Cheers, Mate!")
```

```
Hello, World!
Cheers, Mate!
```



```
name = "ali raza"
print(name)
```

```
ali raza
```

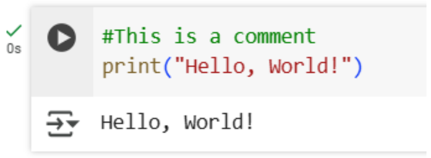


```
if 5 > 2:
  print("Five is greater than two!")
```

```
Five is greater than two!
```

Python Comments

Comments starts with a #, and Python will ignore them:



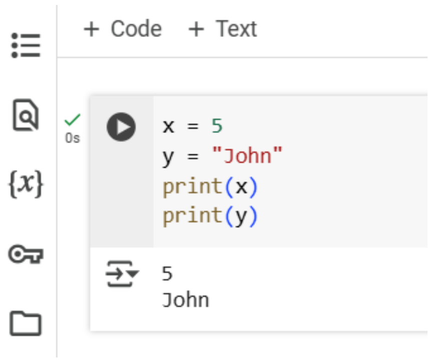
```
#This is a comment
print("Hello, World!")
```

```
Hello, World!
```

Python Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.



```
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```

Built-in Data Types

In programming, data type is an important concept.

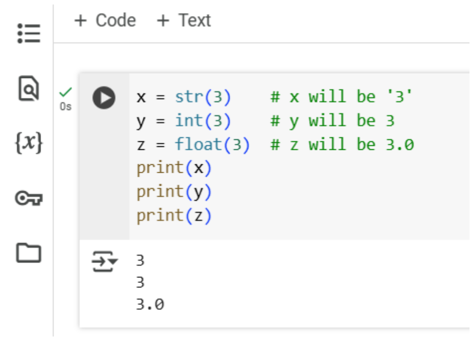
Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

Casting

If you want to specify the data type of a variable, this can be done with casting.

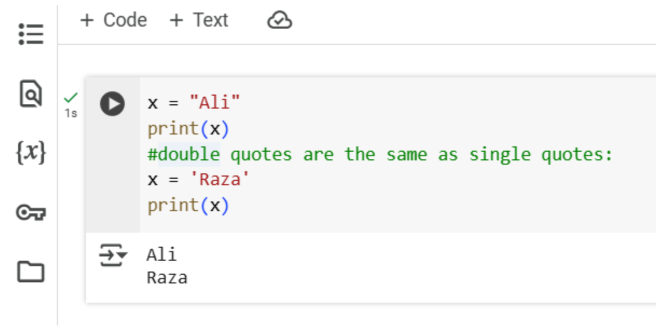


```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
print(x)
print(y)
print(z)
```

```
3
3
3.0
```

Single or Double Quotes?

String variables can be declared either by using single or double quotes:

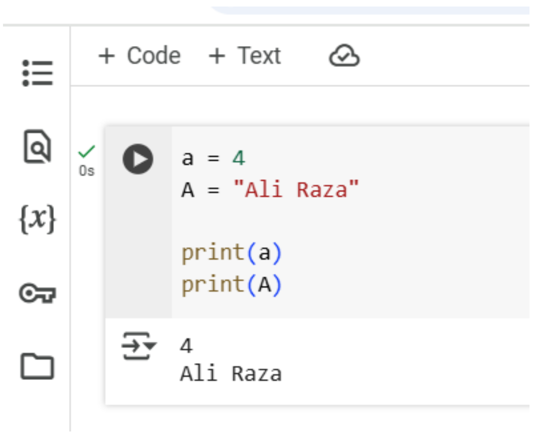


```
x = "Ali"
print(x)
#double quotes are the same as single quotes:
x = 'Raza'
print(x)
```

```
Ali
Raza
```

Case-Sensitive

Variable names are case-sensitive.



```
a = 4
A = "Ali Raza"

print(a)
print(A)
```

```
4
Ali Raza
```

A variable name must start with a letter or the underscore character

A variable name cannot start with a number

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

Variable names are case-sensitive (age, Age and AGE are three different variables)

A variable name cannot be any of the Python keywords.

Multi Words Variable Names

Camel Case

Each word, except the first, starts with a capital letter:

my**V**ariable**N**ame = "John"

Pascal Case

Each word starts with a capital letter:

MyVariableName = "John"

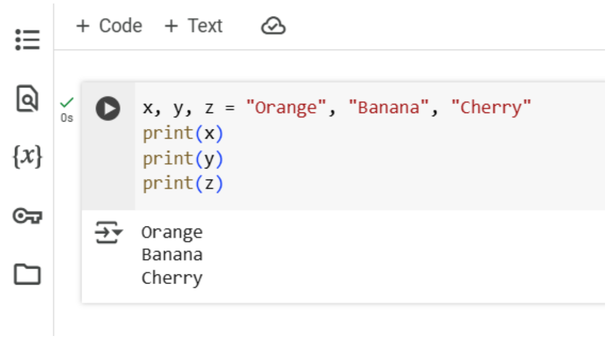
Snake Case

Each word is separated by an underscore character:

my_variable_name = "John"

Many Values to Multiple Variables

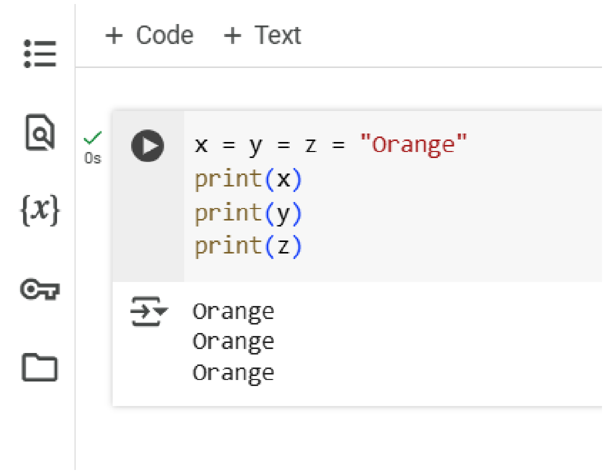
Python allows you to assign values to multiple variables in one line:



```python
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
```

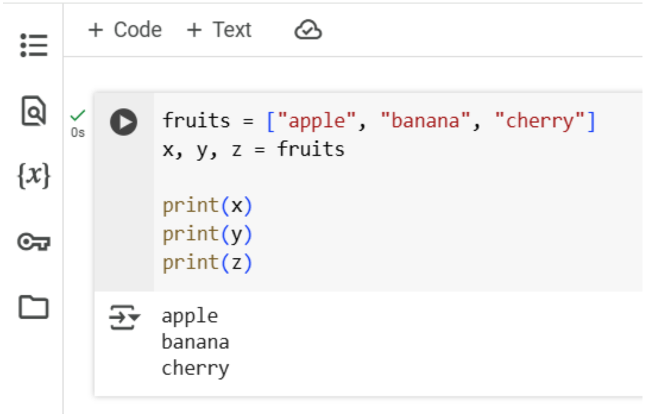
And you can assign the *same* value to multiple variables in one line:



```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

```
Orange
Orange
Orange
```

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

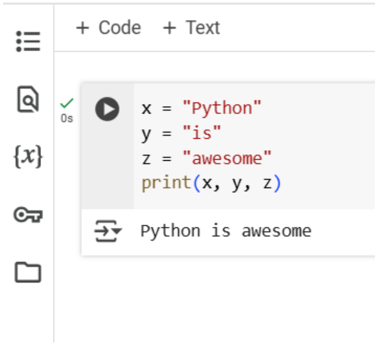


```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits

print(x)
print(y)
print(z)
```

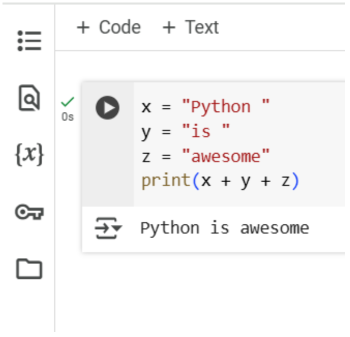
```
apple
banana
cherry
```

Output Variables



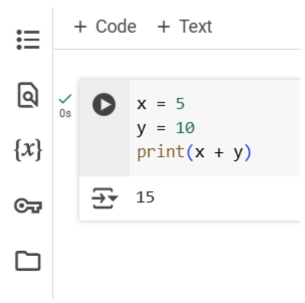
```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

```
Python is awesome
```



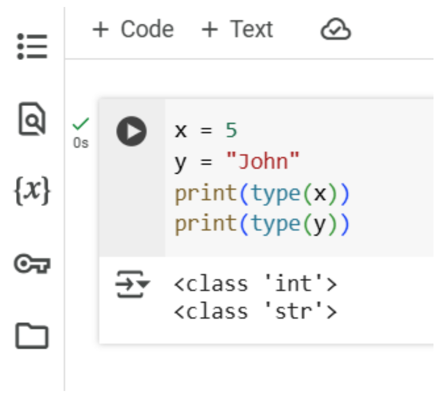
```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

```
Python is awesome
```



```
x = 5
y = 10
print(x + y)
```

```
15
```



```
x = 5
y = "John"
print(type(x))
print(type(y))
```

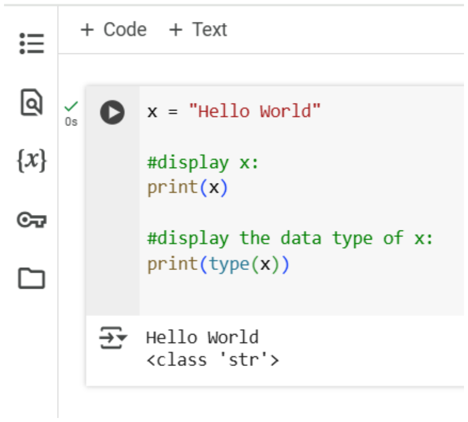
```
<class 'int'>
<class 'str'>
```

You can get the **data type** of a variable with the type() function.

Result defined you the type of x and type of y

In Python, the data type is set when you assign a value to a variable:

Example	Data Type	
x = "Hello World"	Text Type:	str
x = 20	Numeric Types:	int
x = 20.5	Numeric Types:	float
x = 1j	Numeric Types:	complex
x = ["apple", "banana", "cherry"]	Sequence Types:	list
x = ("apple", "banana", "cherry")	Sequence Types:	tuple
x = range(6)	Sequence Types:	range
x = {"name" : "John", "age" : 36}	Mapping Type:	dict
x = {"apple", "banana", "cherry"}	Set Types:	set
x = frozenset({"apple", "banana", "cherry"})	Set Types:	frozenset
x = True	Boolean Type:	bool
x = b"Hello"	Binary Types:	bytes
x = bytearray(5)	Binary Types:	bytearray
x = memoryview(bytes(5))	Binary Types:	memoryview
x = None	None Type:	NoneType

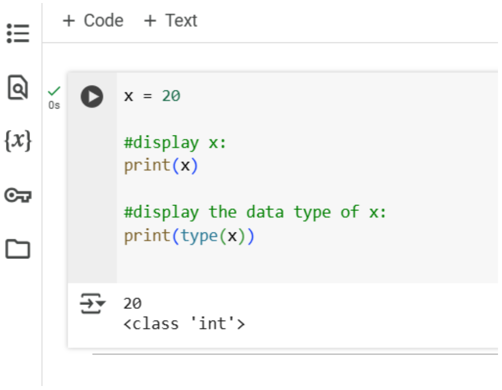


```python
x = "Hello World"

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
Hello World
<class 'str'>
```

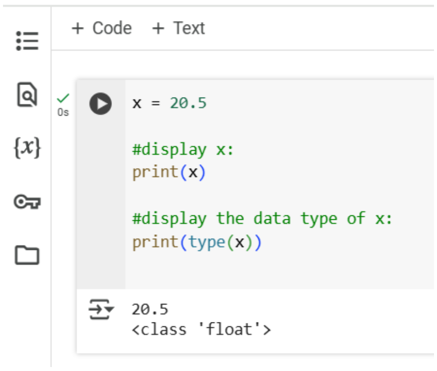


```python
x = 20

#display x:
print(x)

#display the data type of x:
print(type(x))
```

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

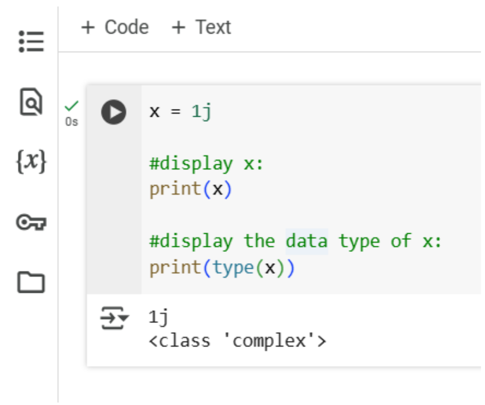


```python
x = 20.5

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
20.5
<class 'float'>
```

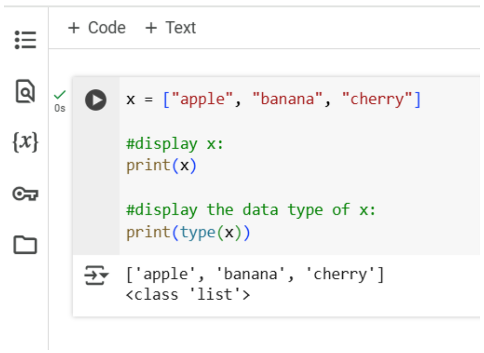


```python
x = 1j

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
1j
<class 'complex'>
```

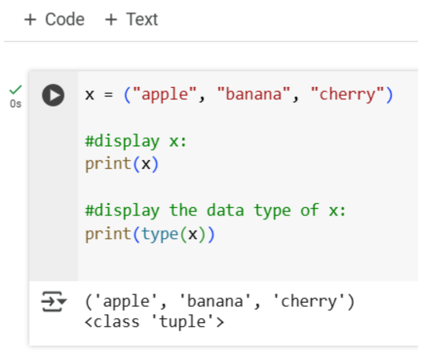


```python
x = ["apple", "banana", "cherry"]

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
['apple', 'banana', 'cherry']
<class 'list'>
```

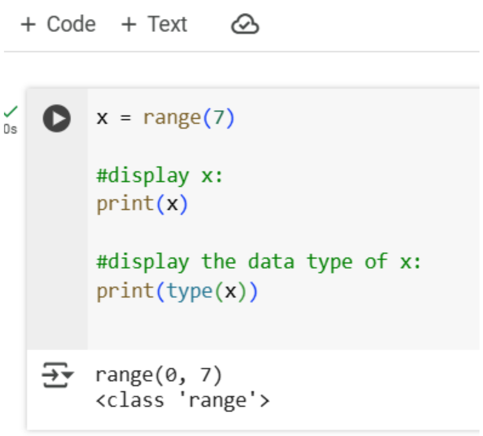


```python
x = ("apple", "banana", "cherry")

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
('apple', 'banana', 'cherry')
<class 'tuple'>
```

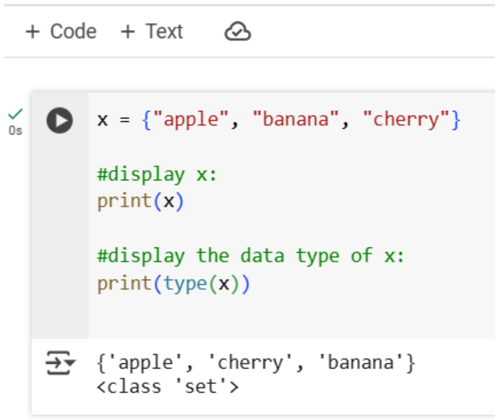


```python
x = range(7)

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
range(0, 7)
<class 'range'>
```

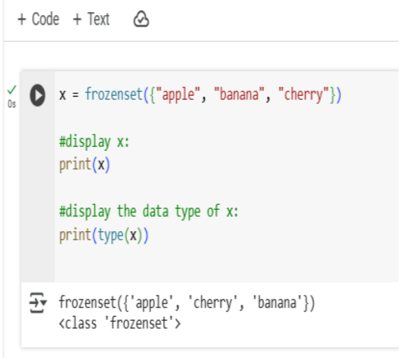


```python
x = {"apple", "banana", "cherry"}

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
{'apple', 'cherry', 'banana'}
<class 'set'>
```

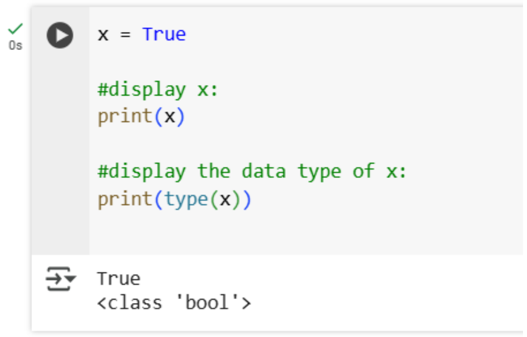


```python
x = frozenset({"apple", "banana", "cherry"})

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
frozenset({'apple', 'cherry', 'banana'})
<class 'frozenset'>
```

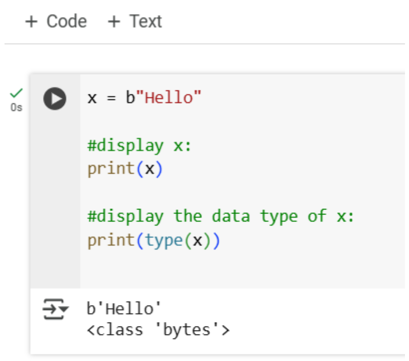


```python
x = True

#display x:
print(x)

#display the data type of x:
print(type(x))
```

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

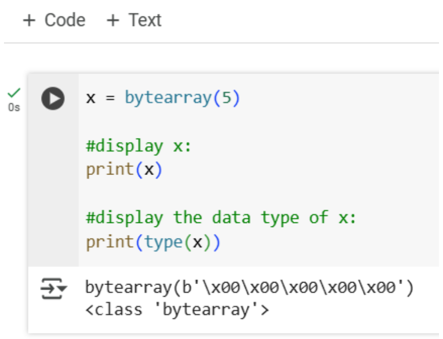


```python
x = b"Hello"

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
b'Hello'
<class 'bytes'>
```



```python
x = bytearray(5)

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
bytearray(b'\x00\x00\x00\x00\x00')
<class 'bytearray'>
```

Python Numbers

Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Float

Float, or **"floating point number"** is a number, positive or negative, containing one or more decimals.

Float can also be scientific numbers with an **"e"** to indicate the power of 10.

Complex

Complex numbers are written with a **"j"** as the imaginary part:

List

Lists are used to store **multiple items** in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using **square brackets:**

Tuple

Tuples are used to store **multiple items** in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is **ordered and unchangeable.**

Tuples are written with **round brackets.**

Range () Function

Create a sequence of numbers from 0 to 5, and print each item in the sequence:

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

range*(start, stop, step*)   x = range(3, 20, 2)

Dict

Create a dictionary containing personal information:

Dictionaries are used to store data values in key: value pairs.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

Dictionaries are written with **curly brackets,** and have keys and values:

Set

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is **unordered, unchangeable*, and unindexed.**

Sets are written with **curly brackets.**

Frozenset () Function

The frozenset() function returns an unchangeable frozenset object (which is like a set object, only unchangeable).

Booleans

Booleans represent one of two values: True or False.

In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

Bytes() Function

The bytes() function returns a bytes object.

It can convert objects into bytes objects, or create empty bytes object of the specified size.

The difference between bytes() and bytearray() is that bytes() returns an object that cannot be modified, and bytearray() returns an object that can be modified.

bytes(*x, encoding, error*)

Bytearray() Function

The bytearray() function returns a bytearray object.

It can convert objects into bytearray objects, or create empty bytearray object of the specified size.

bytearray(*x, encoding, error*)

Python Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

You can display a string literal with the print() function:

print("Hello")

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

print("He is called 'Johnny'")

You can assign a multiline string to a variable by using three quotes:

a = """Lorem ipsum dolor sit amet, consectetur adipiscing elit, """ print(a)

Since strings are arrays, we can loop through the characters in a string, with a for loop.

for x in "banana":   print(x)

To get the length of a string, use the len() function.

a = "Hello, World!" print(len(a))   result 13

To check if a certain phrase or character is present in a string, we can use the keyword in.

txt = "The best things in life are free!"   print("free" in txt)   result   true

txt = "The best things in life are free!"   print("live" in txt) result false

Print only if "free" is present:

txt = "The best things in life are free!"   if "free" in txt: print("Yes, 'free' is present.")

Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

b = "Hello, World!"   print(b[:5])   result hello

Slice From the Start

By leaving out the start index, the range will start at the first character:

b = "Hello, World!"   print(b[:5])   result Hello

Slice To the End

Get the characters from position 2, and all the way to the end:

b = "Hello, World!"   print(b[2:])   result llo, World!

Negative Indexing

Use negative indexes to start the slice from the end of the string:

b = "Hello, World!"   print(b[-5:-2])   result orl

From: "o" in "World!" (position -5)   To, but not included: "d" in "World!" (position -2):