National University of Computer and Emerging Sciences

Lab 01
DL2001-Introduction to Data Science Lab

Course Instructor	Ms. Mariam Nasim
Lab Instructor(s)	Rida Amir
Section	BDS-3A
Semester	Fall 2025

Department of Data Science

FAST-NU, Lahore, Pakistan

Introduction to Python

Objectives:

- Understand the basic environment of Jupyter Notebook.
- Write and execute Python code in cells.
- Use variables and operators effectively.
- Work with basic data types in Python.
- Implement control structures like if, for, and while.

Python Interpreters

1. Google Colab

Google Colab (short for Colaboratory) is a free, cloud-based Jupyter notebook environment provided by Google. It allows you to:

- Write and execute Python code.
- Use GPUs/TPUs for faster computations.
- Collaborate on projects in real time.
- Access powerful libraries without installation.

Step 1: Access Google Colab

1. **Sign in to Google Account:** Ensure you are logged into your Google account (sign in with your FAST LHR email)
2. **Open Google Colab:**
    - Go to: Google Colab.
    - Alternatively, you can access it via Google Drive:
        - Open Google Drive.
        - Click on **"New" > "More" > "Google Colaboratory"** to create a new notebook.

Step 2: Understand the Notebook Interface

The Google Colab interface is similar to Jupyter Notebook. Key components include:

1. **Cells:**
    - **Code Cell:** For writing and running Python code.
    - **Text Cell:** For writing markdown or plain text (e.g., instructions or explanations).
2. **Toolbar:**

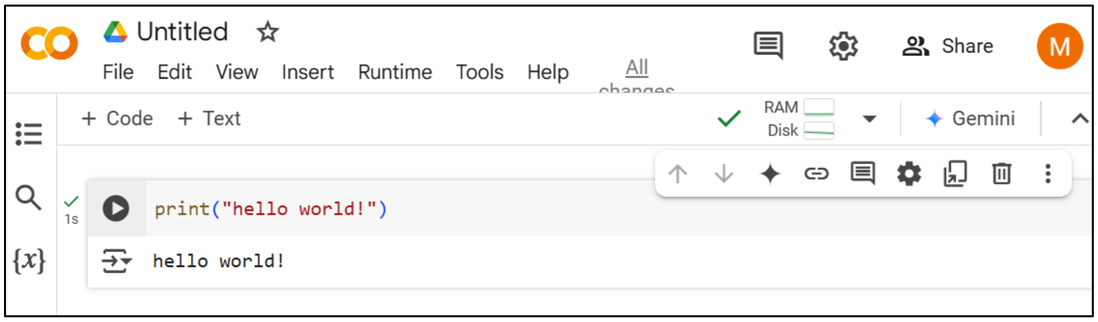
        - **File:** To create, open, save, or download notebooks.
        - **Runtime:** To manage the computational backend (e.g., restart runtime, select GPU).
        - **Insert:** To add new cells (code or text).
    3. **Sidebar:**
        - Manage files and datasets in the notebook environment.

Step 3: Create a New Notebook

1. **Start a New Notebook:**
    - Open Colab and click on **"New Notebook"**.
    - Alternatively, in Google Drive, go to **"New > More > Google Colaboratory"**.
2. **Rename the Notebook:**
    - Click on the notebook's default name (**Untitled.ipynb**) in the top-left corner.
    - Provide a meaningful name for the notebook.

Step 4: Write and Run Code

1. **Add a Code Cell:**
    - Click on the "+ Code" button to add a new cell.
2. **Write Python Code:**
3. **Run the Code:**
    - Click the **Play** button on the left of the cell or press **Shift + Enter**.



```
print("hello world!")
```

```
hello world!
```

2. Jupyter Notebooks:

Step 1: Download Anaconda

1. Visit the official Anaconda website.
2. Download the appropriate version for your operating system (Windows, macOS, or Linux).

    - Choose the **Python 3.x** version.

Step 2: Install Anaconda

Step 3: Launch Jupyter Notebook via Anaconda Navigator

1. Open the **Anaconda Navigator** (search for it in your start menu or applications folder).
2. In Anaconda Navigator, you'll see an option for **Jupyter Notebook**.
3. Click the **Launch** button next to Jupyter Notebook. It will open in your default web browser.

Lab Material:

You can refer to the following links, if any details have been missed in the in manuals.

a) Data Types
    a. Built in types: https://www.w3schools.com/python/python_datatypes.asp
    b. Typecasting: https://www.w3schools.com/python/python_casting.asp

b) Operators https://www.w3schools.com/python/python_operators.asp
    a. Math
    b. Comparison
    c. Boolean

c) If-else conditions: https://www.w3schools.com/python/python_conditions.asp

d) Loops: https://www.w3schools.com/python/python_for_loops.asp

https://www.w3schools.com/python/python_while_loops.asp

Variables

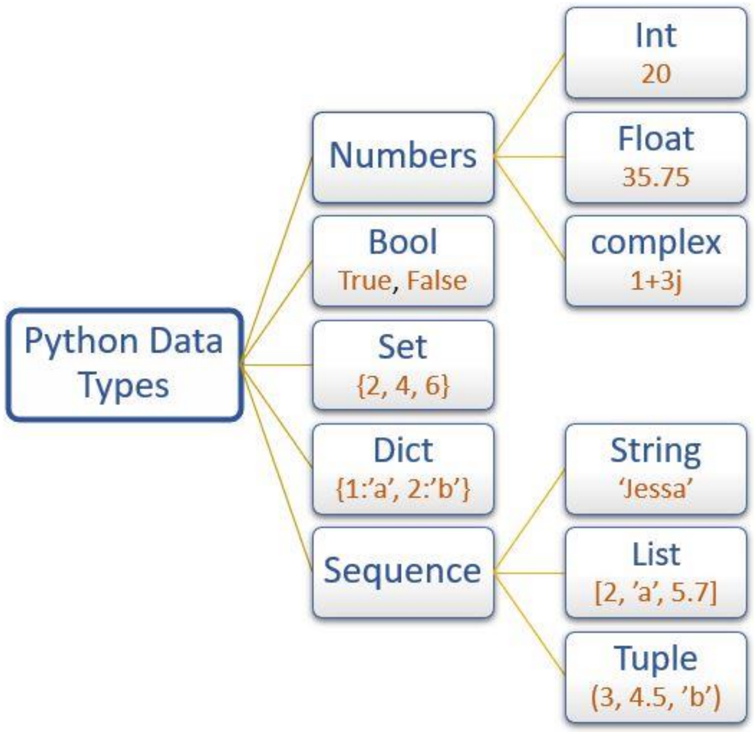
- Variables store data values.
- Python is dynamically typed, meaning no need to declare variable types.
- Names must start with a letter or underscore.
- Cannot use keywords (e.g., if, for, while).

```
x = 10
name = "Alice"
pi = 3.14
```

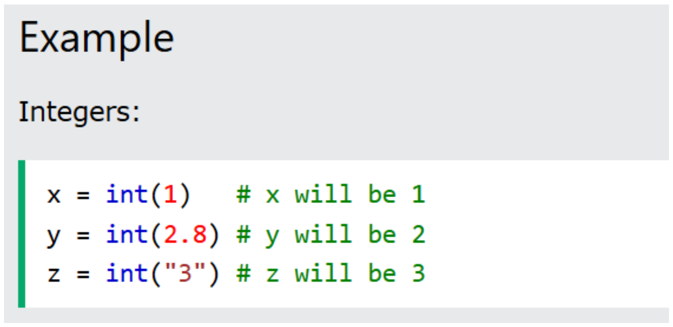
Python Data types:

Python has no command for declaring a variable for any datatype. A variable is created the moment you first assign a value to it. Variable names are casesensitive. Just like in other languages, Python allows you to assign values to multiple variables in one line. Basic data types are:

Data Type	Example	Description
`int`	`5`, `-3`	Whole numbers
`float`	`3.14`, `-2.0`	Decimal numbers
`str`	`"hello"`	Text (string)
`bool`	`True`, `False`	Boolean (logic)
`list`	`[1, 2, 3]`	Ordered collection



The process of explicitly converting the value of one data type (int, str, float, etc.) to another data type is called type casting. In Type Casting, loss of data may occur as we enforce the object to a specific data type.

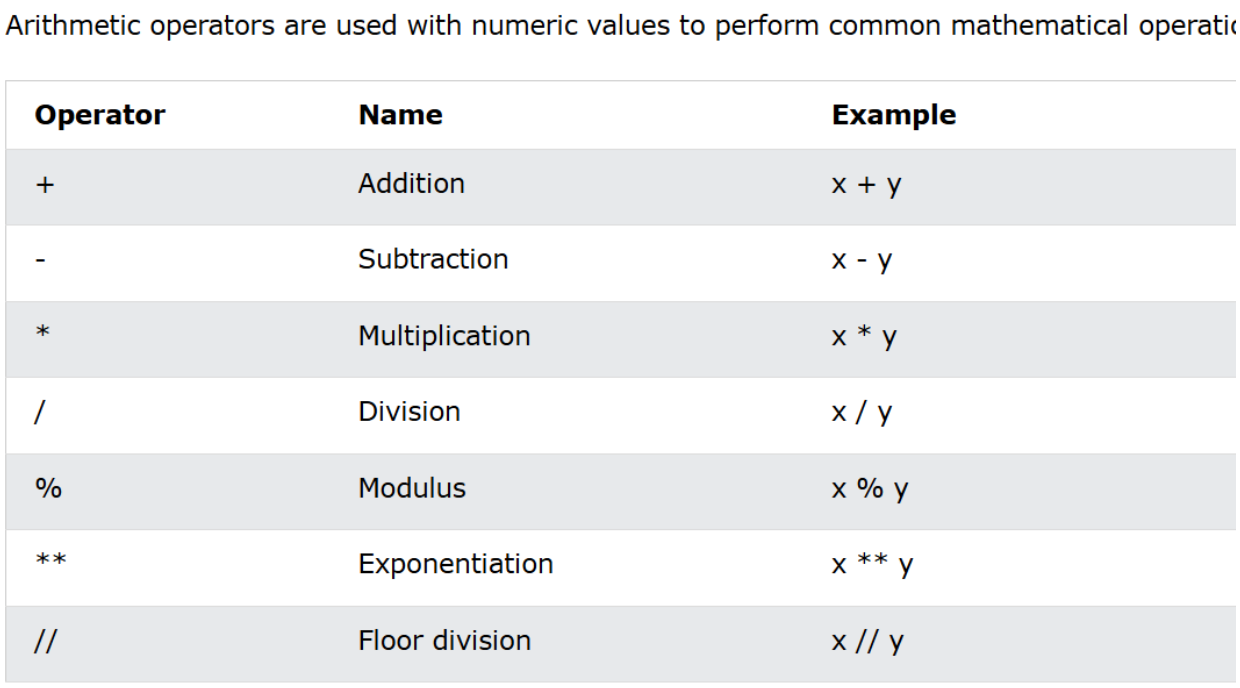

Example

Integers:

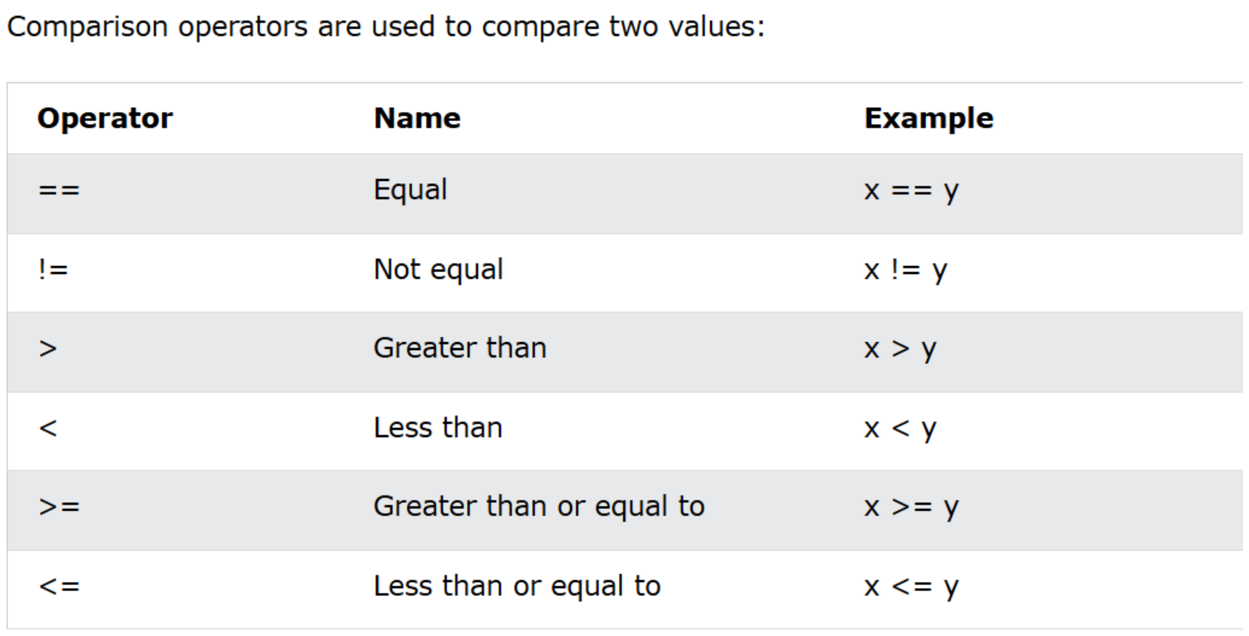
```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

Operators

This section contains the details of different Python operators i.e. Math operators, comparison operators and Boolean operators.

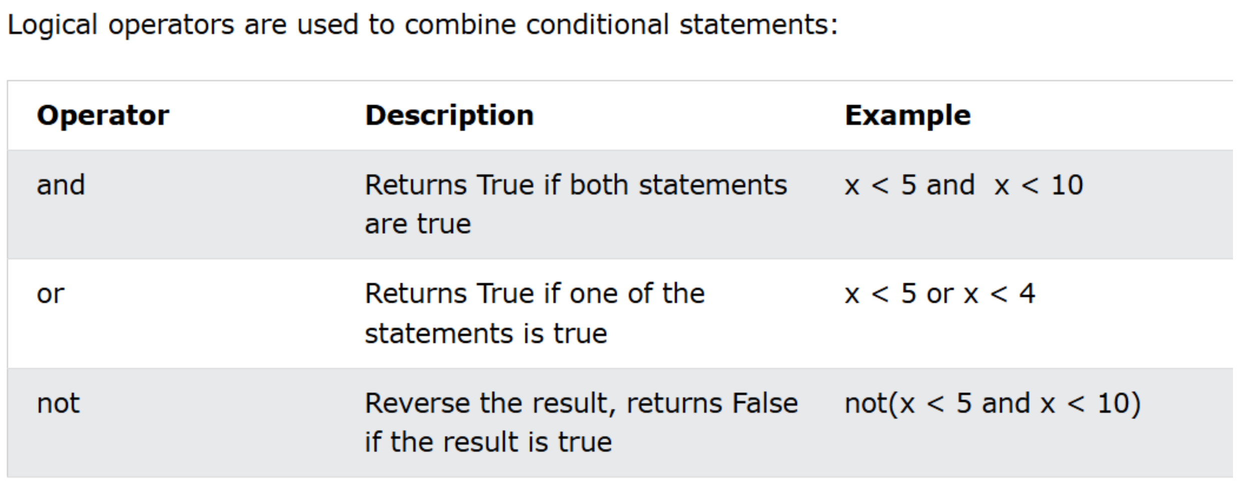

Arithmetic operators are used with numeric values to perform common mathematical operati

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



Comparison operators are used to compare two values:

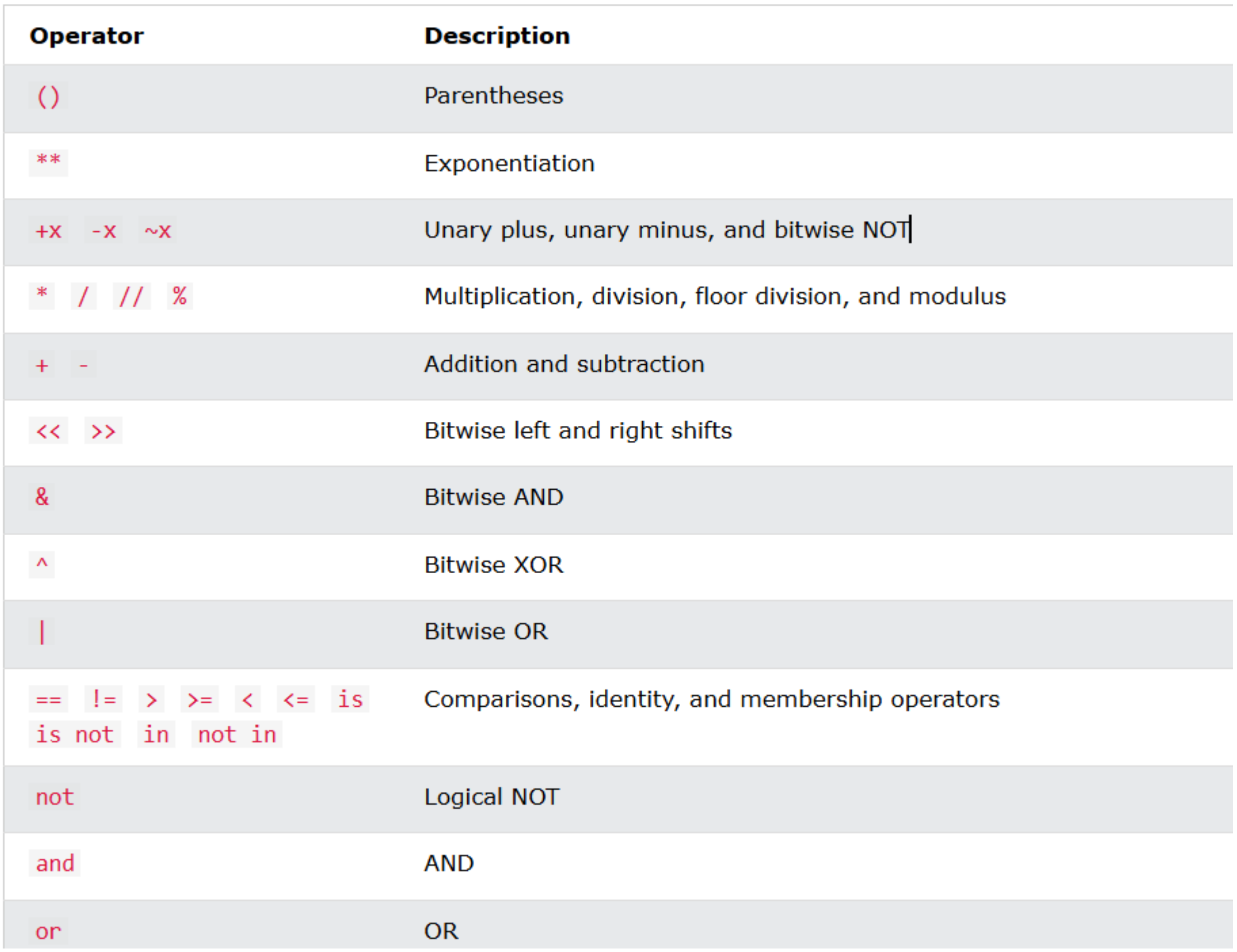
Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y



Logical operators are used to combine conditional statements:

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)

Operator Precedence:



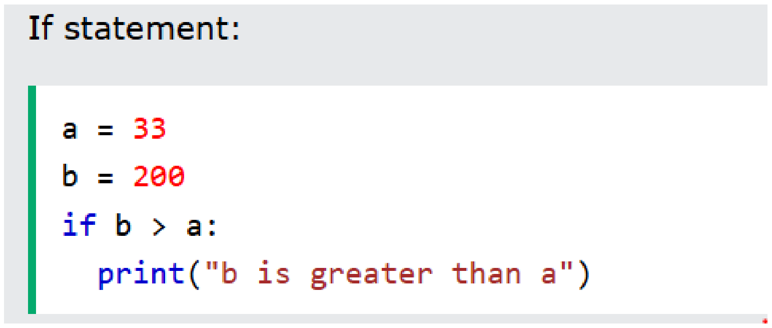
Operator	Description
`()`	Parentheses
`**`	Exponentiation
`+x` `-x` `~x`	Unary plus, unary minus, and bitwise NOT
`*` `/` `//` `%`	Multiplication, division, floor division, and modulus
`+` `-`	Addition and subtraction
`<<` `>>`	Bitwise left and right shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==` `!=` `>` `>=` `<` `<=` `is` `is not` `in` `not in`	Comparisons, identity, and membership operators
`not`	Logical NOT
`and`	AND
`or`	OR

Control Structures:

1) If-else Statement

Python supports conditional statements i.e.if,elif,else. Comparison operators and Boolean operators written in the previous section can be used in if-elif-else statements.

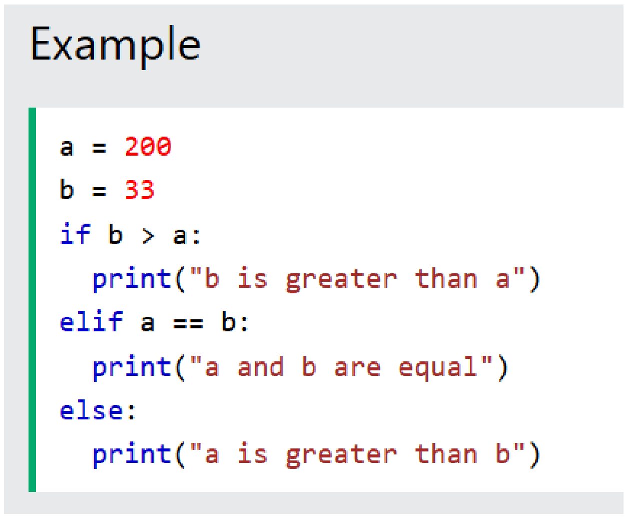
An "if statement" is written by using the if keyword.
Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.


If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

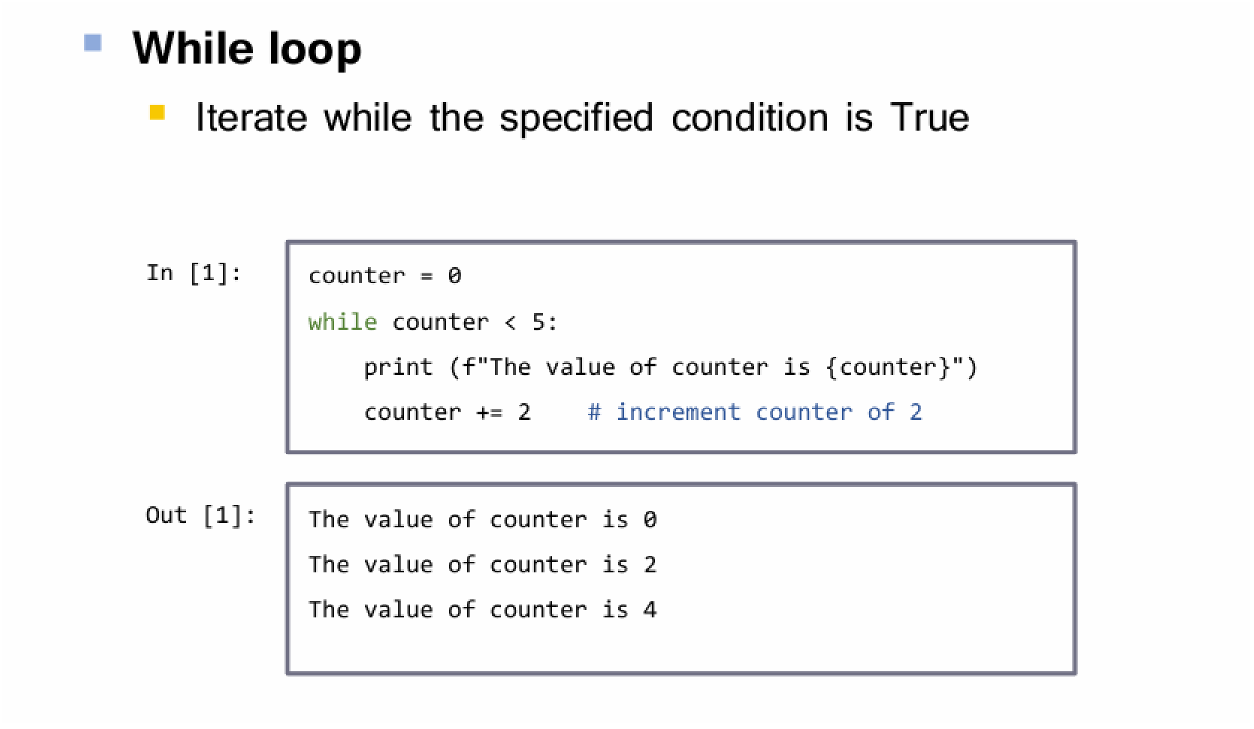
The else keyword catches anything which isn't caught by the preceding conditions.


Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Python conditional statements only require if statement to execute. Both elif and else are optional and used as per requirement.

2) While Loop:



While loop

- Iterate while the specified condition is True

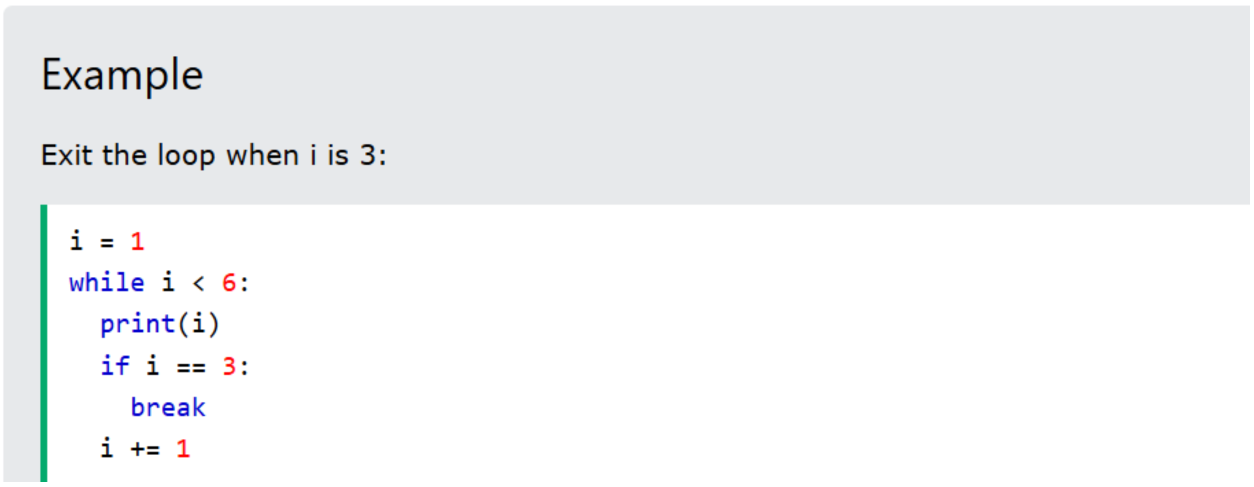
In [1]:

```python
counter = 0
while counter < 5:
    print (f"The value of counter is {counter}")
    counter += 2     # increment counter of 2
```

Out [1]:

```
The value of counter is 0
The value of counter is 2
The value of counter is 4
```

With the `break` statement we can stop the loop even if the while condition is true:

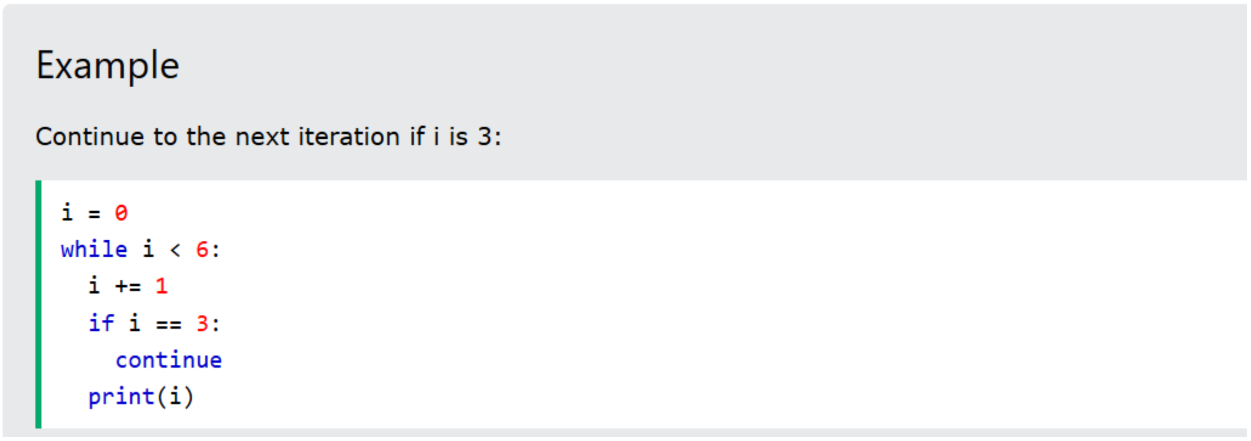


Example

Exit the loop when i is 3:

```python
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

With the `continue` statement we can stop the current iteration, and continue with the next:



Example

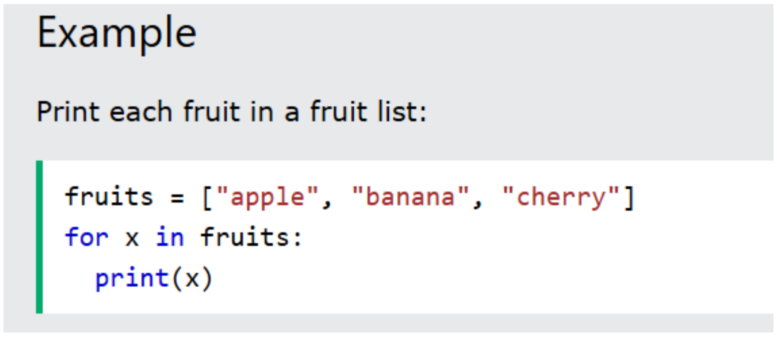
Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

3) For Loop:

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

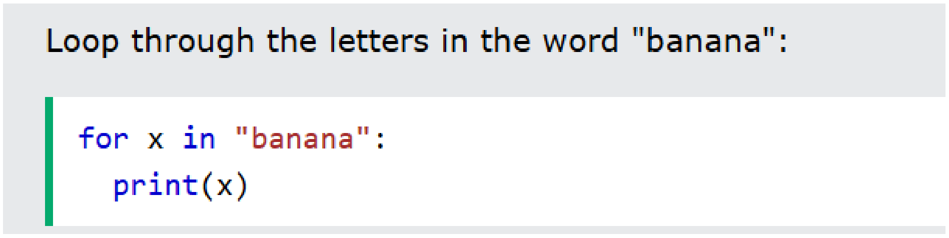
With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.



Example

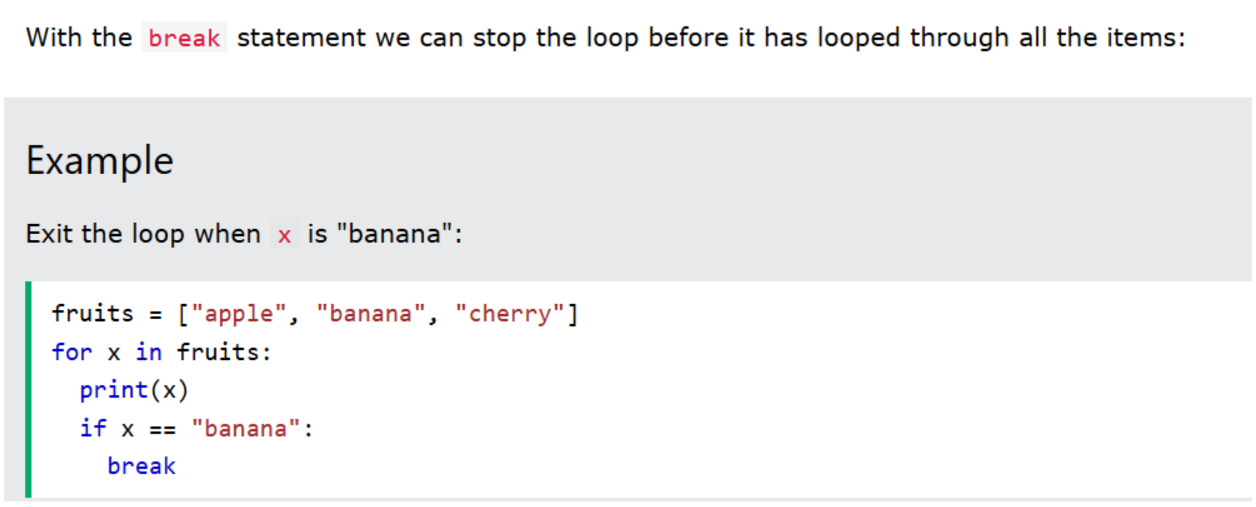
Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```



Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

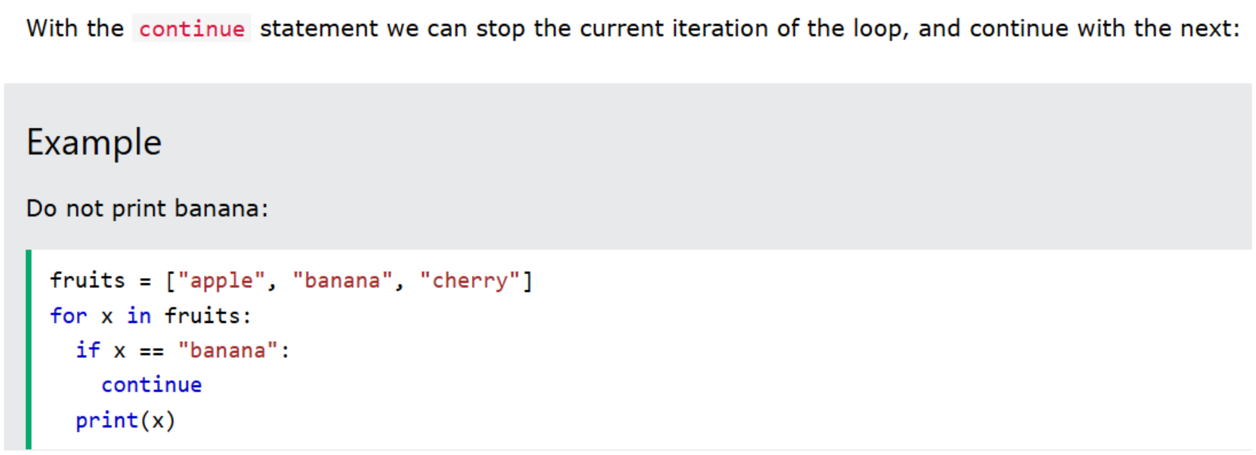


With the break statement we can stop the loop before it has looped through all the items:

Example

Exit the loop when x is "banana":

```python
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```



With the continue statement we can stop the current iteration of the loop, and continue with the next:

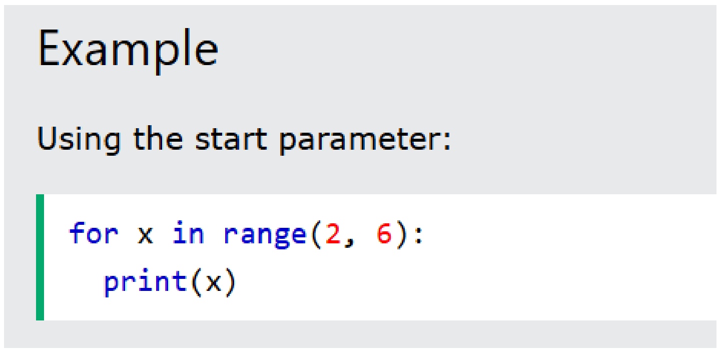
Example

Do not print banana:

```python
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

To loop through a set of code a specified number of times, we can use the range() function,

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



Example

Using the start parameter:

```python
for x in range(2, 6):
  print(x)
```

Lab Tasks: (55)

Task 1: (10)

a) Declare variables named **income** and **bonus_percent**, and initialize them with values 6000 and 15 respectively. Use arithmetic operators (/, %, *, +, -) as appropriate to calculate **15% bonus** of your monthly income and then, print the bonus upto 2 decimal place. **(4)**

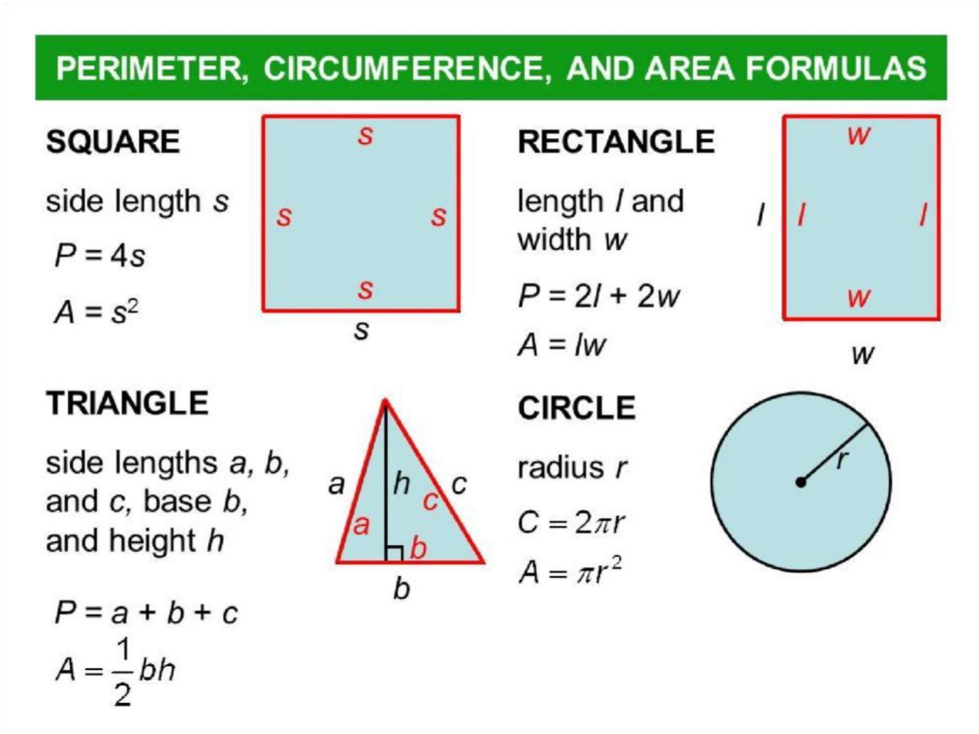
b) Repeat task **(a)**, but this time take both **income** and **bonus_percent** as inputs from the user. **(2)**

c) Store the calculated bonus in a new variable called **bonus**. Create another variable named **total_income**. Add the **bonus** to your **income** and store it in **total_income**. Finally, print the updated income. **(4)**

Task 2: (10)

Take character input from the user as **r or R, t or T or c or C**.
If the user enters r/R then calculate the area and perimeter of Rectangle.
If the user enters t/T then calculate the area and perimeter of Triangle.
If the user enters c/C then calculate the area and perimeter of Circle.
otherwise print the message "The entered value is invalid".

Formulas are given as follow :



Task 3: (10)

Write a python program that **takes two numbers (num1, num2) in input and then** prints numbers starting from num1 till num2 inclusive.
If num1 = 4 and num2 = 15 then your code should Output:

```
4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
```

Make sure num2 is greater than num1, otherwise print appropriate Error message.

Task 4: (10)

Write a program that **reverses a string**. Do not allocate extra space, you must do this reversal by modifying the exiting string in-place with O(1) extra memory.

Example:

```
Input: "hello"
Output: "olleh”
```

Task 5: (10)

Given a strings , determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

Example 1:

```
Input: s = "A man, a plan, a canal: Panama"
Output: true
```

Explanation: "amanaplanacanalpanama" is a palindrome.

Example 2:

```
Input: s = "race a car"
Output: false
```

Explanation: "raceacar" is not a palindrome.

Task 6: (5)

Given a non-negative integer x, compute and return the square root of x. Since the return type is an integer, the decimal digits are truncated, and only the integer part of the result is returned.

Example 1:

```
Input: x = 4
Output: 2
```

Example 2:

```
Input: x = 8
Output: 2
```

Explanation: The square root of 8 is 2.82842..., and since the decimal part is truncated, 2 is returned.

Instructions:

- Submit a **single .ipynb** file. Any other extension would be marked zero.
- **Rename** your file to "rollnum_Name.ipynb" before submitted. Files with no names will be marked zero.
- Any submission **after deadline**, will be marked zero.