National University of Computer and Emerging Sciences

# Artificial Intelligence

**Lab 01**


Fast School of Computing

FAST-NU, Lahore, Pakistan

# Objectives

- Python Integrated Development Environments (IDEs)
- Python data types
- Python operators (math, comparison, Boolean)
- Python condition and loops
- Python functions

# Contents



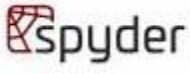
## 1. Python Integrated Development Environments (IDEs)

Here are some popular Python Integrated Development Environments (IDEs) that you can use to write Python code:

**PyCharm**: This is a full-featured IDE that offers many features, including code completion, code navigation, refactoring, and debugging. It is one of the most popular IDEs for experienced Python developers.

**Visual Studio Code**: This is a lightweight, open-source IDE that is gaining popularity among Python developers. It offers many features, including code completion, debugging, and Git integration.

**Spyder**: This is an IDE designed specifically for scientific computing. It offers features such as code completion, debugging, and variable explorer. It is a great choice for data scientists and researchers.



**Jupyter Notebook**: This is a web-based interactive computing environment that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It is widely used in data science and scientific computing.

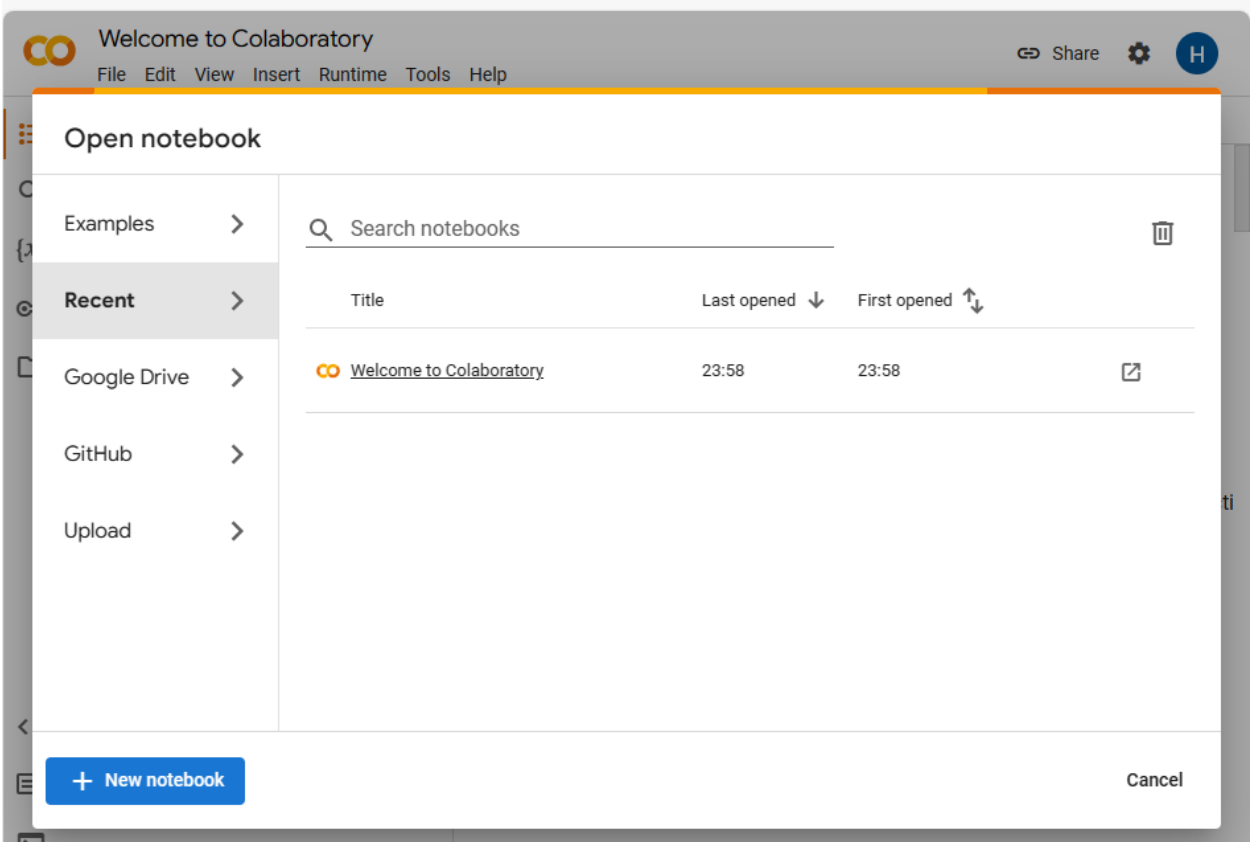
**Anaconda**: This is a distribution of Python and R that includes many popular packages for data science, machine learning, and scientific computing. It also includes the Jupyter Notebook and Spyder IDEs.

**Google Colab**: This is a cloud-based Jupyter notebook environment that allows you to write and execute Python code in your browser, with zero configuration required. You can access GPUs free of charge and easily share your work with others.

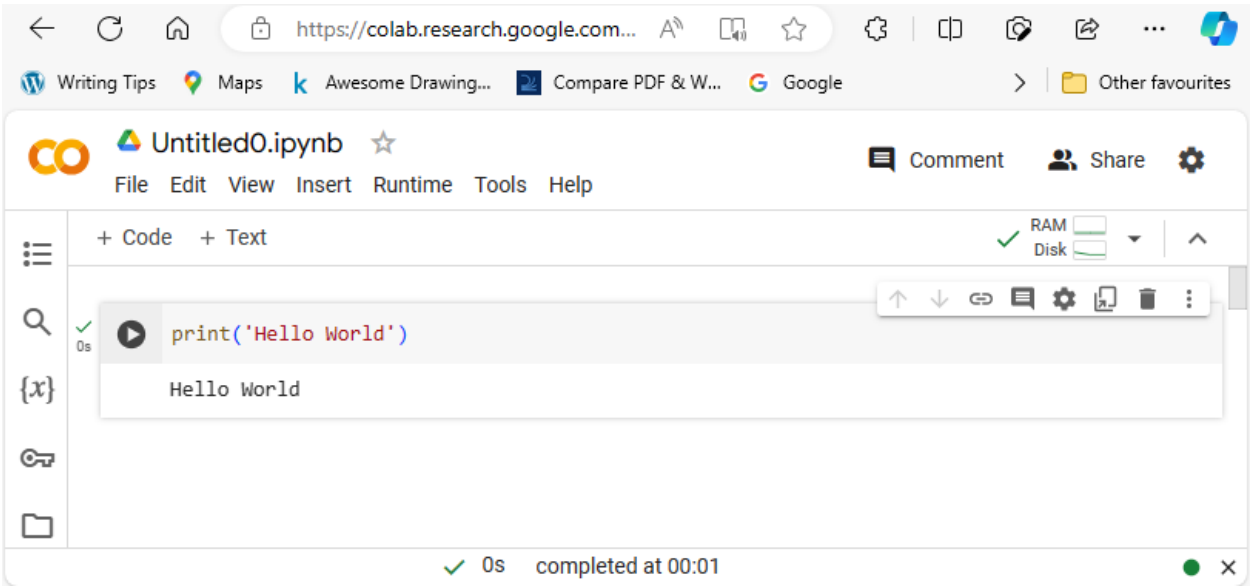
**We'll utilize Google Colab for running Python programs online, for offline usage we will recommend installing PyCharm.**

## 2. “Hello World” in python

1. Visit the following link: https://colab.research.google.com/
2. Sign in using your NU email ID.
3. You will be directed to the ‘Welcome to Colaboratory’ page.
4. Press + New notebook option or Go to File -> New Notebook.




5. A new notebook is created by the name ‘Untitled0.ipynb’.
6. Rename the notebook to your roll number.
7. Write your first Python program by typing the following statement in the first cell:
   `print('Hello World')`
8. Execute this cell by clicking the play button on the left of the cell or by pressing Ctrl+Enter.
9. You will notice that the notebook will connect to a runtime. RAM and disk resources are allocated and our program will display **Hello World** after execution.




# 3. Data Types

The following section describes the standard types that are built into the Python interpreter. These datatypes are divided into different categories like numeric, sequences, mapping etc. Typecasting is also discussed below.

## 3.1 Built-in Types

The following chart summarizes the standard data types that are built into the Python interpreter.

| Sr # | Categories | Data Type | Examples |
|---|---|---|---|
| 1 | Numeric Types | int | -2, -1, 0, 1, 2, 3, 4, 5, int (20) |
| 2 | | float | -1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25, float (20.5) |
| 3 | | complex | 1j, complex(1j) |
| 4 | Text Sequence Type | str | 'a', 'Hello!', str ("Hello World") |
| 5 | Boolean Type | bool | True, False, bool (5) |
| 6 | Sequence Types | list | ["apple", "banana", "cherry"], list (("apple", "banana", "cherry")) |

| | | | |
|---|---|---|---|
| 7 | | tuple | ("apple", "banana", "cherry"), tuple (("apple", "banana", "cherry")) |
| 9 | Mapping Type | dict | {"name": "John", "age": 36},<br>dict (name="John", age=36) |
| 10 | Set Types | set | {"apple", "banana", "cherry"}, set (("apple", "banana", "cherry")) |
| 11 | | frozenset | Frozenset ({"apple", "banana", "cherry"}) |

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 case-sensitive. Just like in other languages, Python allows you to assign values to multiple variables in one line.

## 3.1.1 Dictionaries in Python

A dictionary in Python is defined using curly braces {}. Each element in a dictionary consists of a key-value pair separated by a colon :. Keys are unique and immutable (such as strings, numbers, or tuples), while values can be of any data type.

**Creating a Dictionary:**

```
# Creating an empty dictionary
my_dict = {}
# Creating a dictionary with initial values
my_dict = {'apple': 2, 'banana': 3, 'orange': 4}
```

**Accessing Elements:**

```
# Accessing value using key
print(my_dict['apple'])  # Output: 2
# Using the get() method (returns None if key not found)
print(my_dict.get('banana'))  # Output: 3
```

**Adding and Updating Elements:**

```
# Adding a new key-value pair
my_dict['grape'] = 5

# Updating the value of an existing key
my_dict['apple'] = 6
```

**Removing Elements:**

```
# Removing a key-value pair
del my_dict['orange']
# Using pop() method (returns the value)
```

```
value = my_dict.pop('banana')
```

**Dictionary Methods:**

- **keys()**: Returns a view of all the keys in the dictionary.
- **values()**: Returns a view of all the values in the dictionary.
- **items()**: Returns a view of all key-value pairs in the dictionary.
- **clear()**: Removes all elements from the dictionary.
- **copy()**: Returns a shallow copy of the dictionary.

**Example:**

```
# Example of dictionary methods
print(my_dict.keys())    # Output: dict_keys(['apple', 'banana', 'grape'])
print(my_dict.values())  # Output: dict_values([6, 3, 5])
print(my_dict.items())   # Output: dict_items([('apple', 6), ('banana', 3), ('grape', 5)])
# Iterating through key-value pairs
for key, value in my_dict.items():
   print(key, value)
```

## 3.2 Typecasting

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.

```
# cast to float
x=float(2)
y=float(30.0)
z=float("20")
print(x)
print(y)
print(z)

# cast to str
x=str(2)
y=str(30.0)
z=str("20")
print(x)
print(y)
print(z)
# Sum two numbers using typecast
num_int = 123
num_str = "456"
```

```
print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))
num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))
num_sum = num_int + num_str
print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

Notice the *type()* function used in the above example. Find out what it does. Execute the given example in Jupyter notebook to observe the result of type casting.

# 4. Operators

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

## 4.1 Math Operators

From Highest to Lowest precedence:

| **Operators** | **Operation** | **Example** |
|---|---|---|
| ** | Exponent | 2 ** 3 = 8 |
| % | Modulus/Remainder | 22 % 8 = 6 |
| // | Integer division | 22 // 8 = 2 |
| / | Division | 22 / 8 = 2.75 |
| * | Multiplication | 3 * 3 = 9 |
| - | Subtraction | 5 - 2 = 3 |
| + | Addition | 2 + 2 = 4 |

## 4.2 Comparison Operators

From Highest to Lowest precedence:

| **Operator** | **Meaning** |
|---|---|
| == | Equal to |
| != | Not equal to |

| Operator | Meaning |
|---|---|
| < | Less than |
| > | Greater Than |
| <= | Less than or Equal to |
| >= | Greater than or Equal to |

## 4.3 Boolean Operators

There are three Boolean operators: and, or, and not.

| and Operator | or Operator | not operator |
|---|---|---|
| True and True | True and True | not True |
| True and False | True and False | not False |
| False and True | False and True | |
| False and False | False and False | |

**Green** Expressions evaluate to True whereas **Red** Expressions evaluate to False.

# 5. If-elif-else Conditions

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.

**Note: Python uses indentation instead of curly-brackets to define the scope in the code. Therefore, be cautious while indenting.**

**if Statement Example:**

```
name = 'Alice'
if name == 'Alice':
    print('Hi, Alice.')
```

**if-else Statement Example:**

```
name = 'Bob'
if name == 'Alice':
    print('Hi, Alice.')

else:

    print('Hello, stranger.')
```

**if-elif-else Statement Example:**

```
name = 'Bob'
age = 30
if name == 'Alice':
   print('Hi, Alice.')
elif age < 12:
   print('You are not Alice, kiddo.')
else:
   print('You are neither Alice nor a little kid.')
```

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

# 6. Loops

Python has two types of loops i.e. `while`, `for.` Use Jupyter notebook to execute all code snippets given in the examples below to observe their results.

## 6.1 While Loop Example with break Statement

With the `while` loop we can execute a set of statements as long as a condition is true or the loop execution reaches a `break` statement.

```
while True:
   print('Please type your name.')
   name = input()
   if name == 'your name':
      break
print('Thank you!'
```

`input()` in the above example is a built-in Python function which is discussed in the functions section below.

## 6.2 While Loop Example with continue Statement

When the program reaches a `continue` statement, the program execution immediately jumps back to the start of the loop.

```
while True:
   print('Who are you?')
   name = input()
   if name != 'Joe':
      continue
   print('Hello, Joe. What is the password? (It is a fish.)')
   password = input()
   if password == 'swordfish':
      break
print('Access granted.')
```

## 6.3 for Loop Example with range ()

```
print('My name is')
for i in range(5):
   print('Jimmy Five Times ({})'.format(str(i)))
```

## 6.4 for Loop Example with range () arguments

The `range()` function can also be called with three arguments. The first two arguments will be the start and stop values, and the third will be the step argument. The step is the amount that the variable is increased by after each iteration.

```
for i in range(0, 10, 2):
   print(i)
```

# 7. Functions

Programmers can define their own functions in Python. Functions can contain all types of Python statements like variables, conditions and loops etc.

A function in Python starts with `def` keyword followed by the function name with round brackets. Function parameters can be passed depending on the requirement.

```
def hello(name):
    print('Hello {}'.format(name))
hello('Alice') #Hello Alice
hello('Bob') #Hello Bob
```

### 7.1 Function Example with Return Statement

A return statement consists of the following:

- The return keyword.
- The value or expression that the function should return.

```
import random #Syntax to import Python libraries
def getAnswer (answerNumber):
   if answerNumber == 1:
```

```
        return 'It is certain'
    elif answerNumber == 2:
        return 'It is decidedly so'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Reply hazy try again'
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Concentrate and ask again'
    elif answerNumber == 7:
        return 'My reply is no'
    elif answerNumber == 8:
        return 'Outlook not so good'
    elif answerNumber == 9:
        return 'Very doubtful'
r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

## 7.2 Built-in Functions

The Python interpreter has a number of functions built into it that are always available. We have already covered a few built-in functions in the datatypes section above. Refer to this link for the complete list of Python built-in functions.

Execute the code given below in your Jupyter notebook to find the results of built-in functions.

```
# abs integer number
num = -5
print('Absolute value of -5 is:', abs(num))
# Notice print here, it is also a built-in function
# abs floating number
fnum = -1.45
print('Absolute value of 1.45 is:', abs(fnum))
# input function
x = input('Enter your name:')
print('Hello, ' + x)
# max function
number = [3, 2, 8, 5, 10, 6]
largest_number = max(number);
print("The largest number is:", largest_number)
# print usage
print('Hands-on','python','programming','lab',sep='\n')
# sum function
```

```
my_list = [1,3,5,2,4]
print "The sum of my_list is", sum(my_list)
```

# 8. Exercises

## 8.1 Activity 1

Write a function that prints the egg-timer shape shown below. You may use output statements to print a single asterisk (*), a single hash (#), a single space, or a single newline character. Maximize your use of iteration (with nested for statements) and minimize the number of print statements.

```
Enter Size: 6
##############
#************#
# ********** #
#  ********  #
#   ******   #
#    ****    #
#     **     #
#     **     #
#    ****    #
#   ******   #
#  ********  #
# ********** #
#************#
##############
```

```
Enter Size: 10
######################
#********************#
# ****************** #
#  ****************  #
#   **************   #
#    ************    #
#     **********     #
#      ********      #
#       ******       #
#        ****        #
#         **         #
#         **         #
#        ****        #
#       ******       #
#      ********      #
#     **********     #
#    ************    #
#   **************   #
#  ****************  #
# ****************** #
#********************#
######################
```

## 8.2 Activity 2

Write the code to create a dictionary named “squares”. Each key should be a number between one and ten, and each value should be the square of the number.

## 8.3 Activity 3

Write a Python program that manages student exam scores using a dictionary. The program should allow users to:

1. **Add Student Scores:**
   - Enable users to input student names and their corresponding exam scores to populate the dictionary.
2. **Display Maximum and Minimum Scores:**
   - Implement functions to find and display the names of students with the highest and lowest exam scores, respectively.
3. **Display All Scores:**
   - Provide an option to display all student names and their exam scores.

Ensure that your program handles scenarios where the dictionary is empty.

Your solution should include:

- A brief explanation of the program's functionality.
- Python code implementing the program with clear comments.
- Sample input and output illustrating the program's behavior.

## 8.4 Activity 4

With the help of dictionaries. Develop a program that takes continuous input from the user, consisting of alphabets and numbers. The program is designed to calculate the frequency of digits (0-9) entered by the user and subsequently display the frequency in the form of a horizontal bar chart. An example of the expected output is provided below:

```
Enter Digits. Press Enter to end:
567585895645468758578799560099008989l123213231
Digit   Frequency       Bar Graph
0:              4       ****
1:              4       ****
2:              3       ***
3:              3       ***
4:              2       **
5:              8       ********
6:              4       ****
7:              4       ****
8:              7       *******
9:              7       *******
```

## 8.5 Activity 5

Design a simple Patient Management System in Python using dictionaries to manage patient records at a hospital. The system should provide the following basic CRUD (Create, Remove, Update, Display) functionalities:

1. **Add Patient:**
    - Allow hospital staff to add new patients to the system.
    - Each patient record should include a unique patient ID, name, age, gender, and contact information.
2. **Update Patient Information:**
    - Enable staff to update patient information such as name, age, gender, and contact details.
    - Ensure that changes are reflected in the patient records.
3. **Remove Patient Record:**
    - Implement the ability to remove patient records from the system.
    - Prompt the user for confirmation before removing a patient record.
4. **View All Patients:**
    - Allow staff to view a list of all patients currently registered in the system.
    - Display patient details including patient ID, name, age, gender, and contact information.

Your Patient Management System should be user-friendly, robust, and well-structured. Use dictionaries to store patient records and implement error handling mechanisms to manage invalid inputs or edge cases.

Your solution should include:

- A clear explanation of the program's logic and functionality.
- Python code implementing the simplified Patient Management System using dictionaries.
- Sample input and output demonstrating the CRUD operations.

**Note:** You may assume that there is no need for authentication or security features in this simplified system.

# REFERENCES


- https://docs.python.org/3/library/functions.html
- https://colab.research.google.com
- https://jupyter.org
- https://www.anaconda.com