PYTHON

What is python?

- Python is high-level programming language.
- Python is an interpreted.
- Python is object-oriented.
- Python can connect to database systems. It can also read and modify files.
- 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 IDE(code editors)

- An IDE (Integrated Development Environment) understand your code much better than a text editor.
- Online Compiler from Programiz
- IDLE
- Sublime Text 3
- Atom
- PyCharm
- VScode
- Spyder

Example

Print and input function

- Print()
- Input()

Comments

- # for one line
- “”” comment “””

Variables and Datatypes

- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.
- Type() to get the data type of a variable.
- Casting (int(), str(), float(), bool(),etc…)
- Case-Sensitive :- variable names are case-sensitive.

Operators

- Arithmetic operators(**+**, **-**, *****, **/**, **%**, ******, **//**)
- Assignment operators (=, **+=**, **-=**, ***=**, **/=**, **%=**, ****=**, **//=**)
- Comparison operators (==, !=, >, <, >=, <=)
- Logical operators (and, or, not)

Strings

- One line string :- X = “one line str ” or ‘ one line str ’
- Multiline string :- X = “”” multiline str “”” or ‘’’ ‘’’
- Strings are arrays Ex:- x[index]
- Len(x) to get the length of a string
- Check String :- To check if a certain phrase or character is present in a string, we can use the keyword “in” or “not in”

If statements

- Using “if” keyboard.
- If statement:

  print()
- Using “elif” keyboard.
- The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".
- Using “else” keyboard.
- The else keyword catches anything which isn't caught by the preceding conditions.

If statements

- **Short Hand if**
- if a > b: print("a is greater than b")
- **Short Hand If ... Else**
- print("A") if a > b else print("B")
- print("A") if a > b else print("=") if a == b else print("B")
- Use **logical operator.**
- Nested If
- You can have if statements inside if statements, this is called nested if statements.

If statements

- The **pass** Statement
- if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

Loops

- The **while** Loop
- With the while loop we can execute a set of statements as long as a condition is true.
- ```
  i = 1
  while i < 6:
    print(i)
    i += 1
  ```
  ```

Loops

- Break , continue statement
- **break** statement we can stop the loop even if the while condition is true.
- **continue** statement we can stop the current iteration, and continue with the next.
- Else statement
- **else** statement we can run a block of code once when the condition no longer is true:

Loops

- **For** loops.
- for x in list:
  print(x)
- The **range()** Function
- 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.
- **Nested Loops**
- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":

List

- **Lists** are used to store multiple items in a single variable.
- **Lists** are built-in data types in Python used to store collections of data.
- **Lists** are created using square brackets.
- **Lists** are ordered, to use index to access item.
- **List** items is not unique.
- **List** can have different data types.
- **List** are Changeable (Add, remove, edit)
- Using type(), len() function.

List methods

list=[“ali”,”ahmed”,”adel”,”ali”]

append() Ex: list.append(“hossam”)	Adds an element at the end of the list
clear() Ex: list.clear()	Removes all the elements from the list
copy() Ex: x= list.copy()	Returns a copy of the list
count() Ex: print(list.count(“ali”))	Returns the number of elements with the specified value
index() Ex: print(list.index(“adel"))	Returns the index of the first element with the specified value
insert() Ex: list.insert(1,"hossam")	Adds an element at the specified position
pop() Ex: list.pop(1)	Removes the element at the specified position
remove() Ex: list.remove("ali")	Removes the first item with the specified value
reverse() Ex: list.reverse()	Reverses the order of the list
sort() Ex: list.sort()	Sorts the list

Arrays

- Python does not have built-in support for Arrays.
- to work with arrays in Python you will have to import a library, like the NumPy library or array library.

Tuples

- Tuples are used to store multiple items in a single variable.
- Tuple is built-in data types in Python used to store collections of data.
- A tuple is a collection which is ordered and **unchangeable**.
- Tuples are written with round brackets.
- Using type(), len() function.

Change Tuple Values

- Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.
- But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.
- To join two or more tuples you can use the + operator:

Tuple Methods

tuple=(“ali”,”ahmed”,”adel”,”ali”)

count() Ex: print(tuple.count(“ali”))	Returns the number of elements with the specified value
index() Ex: print(tuple.index(“adel"))	Returns the index of the first element with the specified value

Dictionary

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered*, changeable and does not allow duplicates.
- Dictionaries are written with curly brackets, and have keys and values ‘:’
- Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

Accessing Items

- You can access the items of a dictionary by referring to its key name, inside square brackets.
- There is also a method called `get()` that will give you the same result:

Change/Add Values

- You can change the value of a specific item by referring to its key name:
- The `update()` method will update the dictionary with the items from the given argument.
- The argument must be a dictionary, or an iterable object with key:value pairs.

Rempve items

- There are several methods to remove items from a dictionary:
- The pop() method removes the item with the specified key name.
- The popitem() method removes the last inserted item.

Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- To create a function In Python a function is defined using the **def** keyword.
- To call a function, use the function name followed by parenthesis “ fun_name() ”.
- Using parameter in function fun_name(Parameter,…).
- If we call the function without argument, it uses the default value.
- To let a function return a value, use the `return` statement:

Database

Test MySQL Connector

- To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content:

```
import mysql.connector
```

- If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.

Create Connection

- Start by creating a connection to the database.
- Use the username and password from your MySQL database:

```
db=mysql.connector.connect(
        host="localhost",
        user = "root",
        password = "root",
        )

print("connection created")
```

Create Database

- To create a database in MySQL, use the "CREATE DATABASE" statement:

```
import mysql.connector

db=mysql.connector.connect(
        host="localhost",
        user = "root",
        password = "root",
        )

mycursor = db.cursor()

mycursor.execute("CREATE DATABASE mydatabase")

print("Database created")
```

- If the above code was executed with no errors, you have successfully created a database.

Check if Database Exists

- You can check if a database exist by listing all databases in your system by using the "SHOW DATABASES" statement:

```
db=mysql.connector.connect(
        host="localhost",
        user = "root",
        password = "root",
        )

mycursor = db.cursor()

mycursor.execute("SHOW DATABASES")

for i in mycursor:
    print(i)
```

- Or you can try to access the database when making the connection:
- If the database does not exist, you will get an error.

```
db=mysql.connector.connect(
        host="localhost",
        user = "root",
        password = "root",
        database="mydatabase"
        )
```

Creating a Table

- To create a table in MySQL, use the "CREATE TABLE" statement.
- Make sure you define the name of the database when you create the connection.

```
db=mysql.connector.connect(
        host="localhost",
        user = "root",
        password = "root",
        database="mydatabase"
        )

mycursor = db.cursor()

mycursor.execute("CREATE TABLE customers ( name VARCHAR(50), address VARCHAR(100) )")
```

Check if Table Exists

- You can check if a table exist by listing all tables in your database with the "SHOW TABLES" statement:

```
db=mysql.connector.connect(
        host="localhost",
        user = "root",
        password = "root",
        database="mydatabase"
        )

mycursor = db.cursor()

mycursor.execute("SHOW TABLES")

for i in mycursor:
    print(i)
```

Primary Key

- When creating a table, you should also create a column with a unique key for each record.
- This can be done by defining a PRIMARY KEY.
- We use the statement "INT AUTO_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

```
db=mysql.connector.connect(
        host="localhost",
        user = "root",
        password = "root",
        database="mydatabase"
        )

mycursor = db.cursor()

mycursor.execute("CREATE TABLE clients (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))")
```

Alter Table

- If the table already exists, use the ALTER TABLE keyword:

```
db=mysql.connector.connect(
        host="localhost",
        user = "root",
        password = "root",
        database="mydatabase"
        )

mycursor = db.cursor()

mycursor.execute("ALTER TABLE customers ADD COLUMN id INT AUTO_INCREMENT PRIMARY KEY")
```

Delete a Table

- You can delete an existing table by using the "DROP TABLE" statement:

```
db=mysql.connector.connect(
        host="localhost",
        user = "root",
        password = "root",
        database="mydatabase"
        )

mycursor = db.cursor()

mycursor.execute("DROP TABLE customers")
```

Delete a Database

- You can delete an existing database by using the "DROP DATABASE" statement:

```
db=mysql.connector.connect(
        host="localhost",
        user = "root",
        password = "root",
        database="mydatabase"
        )

mycursor = db.cursor()

mycursor.execute("DROP DATABASE mydatabase")
```

Insert Into Table

- To fill a table in MySQL, use the "INSERT INTO" statement.

```
import mysql.connector

mycon = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="mydatabase"
)

mycursor = mycon.cursor()

#insert data in table
mycursor.execute("INSERT INTO clients (name,address) VALUES('adel','desouk')")

mycon.commit()
```

```
sql = "INSERT INTO clients (name, address) VALUES (%s, %s)"
val = ("Ahmed", "Kafr elshikh")
mycursor.execute(sql, val)

mycon.commit()
```

- Important!: Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

Insert Multiple Rows

- To insert multiple rows into a table, use the executemany() method.
- The second parameter of the executemany() method is a list of tuples, containing the data you want to insert:

```
sql = "INSERT INTO clients (name, address) VALUES (%s, %s)"
val = [
  ('Peter', 'Lowstreet 4'),
  ('Amy', 'Apple st 652'),
  ('Hannah', 'Mountain 21'),
  ('Michael', 'Valley 345'),
  ('Sandy', 'Ocean blvd 2')
]

mycursor.executemany(sql, val)

mycon.commit()
```

Select From a Table

- To select from a table in MySQL, use the "SELECT" statement:
- Select all records from the "customers" table, and display the result:

```
mycursor.execute("SELECT * FROM clients")

result = mycursor.fetchall()

for x in result:
    print(x)
```

- We use the fetchall() method, which fetches all rows from the last executed statement.

Get first row from result

- Using the fetchone() Method
- The fetchone() method will return the first row of the result:

```
mycursor.execute("SELECT * FROM clients")

result = mycursor.fetchone()
print(result)
```

Selecting Columns

- To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

```
mycursor.execute("SELECT id,name FROM clients")

result = mycursor.fetchall()

for x in result:
    print(x)
```

Select With a Filter

- When selecting records from a table, you can filter the selection by using the "WHERE" statement:

```
#get row with id equal 6 (condition)
mycursor.execute("SELECT * FROM clients where id=6")

result = mycursor.fetchall()

for x in result:
    print(x)
```

Sort the Result

- Use the ORDER BY statement to sort the result in ascending or descending order.
- The ORDER BY keyword sorts the result ascending by default. To sort the result in descending order, use the DESC keyword.

```
#sort the result by specific column
mycursor.execute("SELECT * FROM clients ORDER BY name ")

result = mycursor.fetchall()

for x in result:
    print(x)
```

```
#sort the result by specific column descending order
mycursor.execute("SELECT * FROM clients ORDER BY name DESC")

result = mycursor.fetchall()

for x in result:
    print(x)
```

Limit the Result

Start from first position

- You can limit the number of records returned from the query, by using the "LIMIT" statement:

```
mycursor.execute("SELECT * FROM clients LIMIT 5")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Start from Another position

- If you want to return five records, starting from the third record, you can use the "OFFSET" keyword:

```
mycursor.execute("SELECT * FROM clients LIMIT 5 OFFSET 2")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```