# NETW908: Data Engineering

# Lab 1

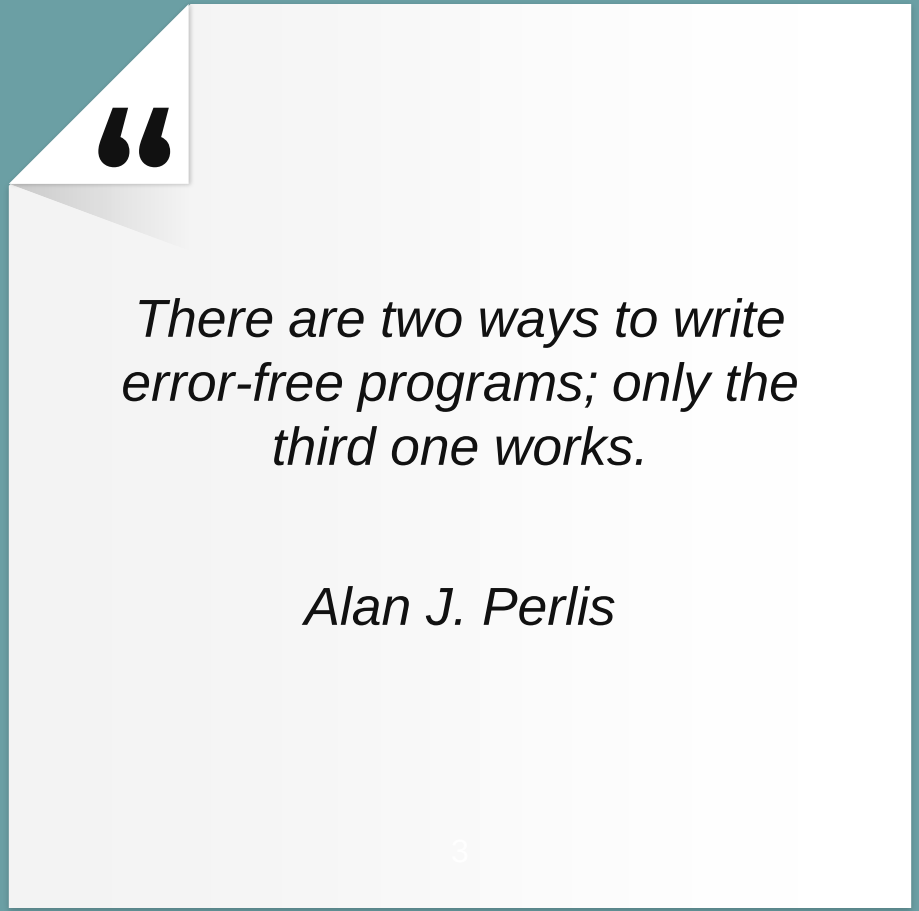
By: Mariham Wasfy

# Hello!

## I am Mariham Wasfy

E-mail: mariham.Ibrahim@guc.edu.eg

Office: C3.307 (inside)


“
There are two ways to write error-free programs; only the third one works.
Alan J. Perlis

# Outline

- Introduction To Python
- Basic Syntax and Importing Libraries
- Error Handling
- Handling Conditions and Loops
- Functions and Classes Definition


# Introduction to Python

1

# 1. Introduction To Python

**What is Python?**

Flexible programming language designed to be human readable.

**Why use Python?**

- Great starter language.
- Great advanced language.
- Wonderful Community.

# 1. Introduction To Python

**What can I build with Python?**

- Graphical user interfaces
- Databases access
- Machine Learning models
- AI projects
- Web App
- Automation utilities
- Anything

**What do I need to get python?**

- Interpreter
- Editor

# 1. Introduction To Python

**Python Features:**

- No compiling or linking
- Rapid development cycle
- No type declaration
- Simpler, shorter and more flexible
- Automatic memory management
- Garbage collection
- High level data types and operations
- Fast development
- Object-oriented programming
- Mixed language systems
- Classes, modules, excepting and multithreading


# Basic Syntax and Importing Libraries

2

# 2. Basic Syntax and Importing Libraries

**Print**

- Displays output to your console.
- Used in debugging.

**Syntax**

```
print('Hello World')
print("Hello World")
print('This is 1st sentence \n This is 2nd sentence')
print('What\'s your name?')
```



You can use either single quote or double quote but stick to your choice for every declaration the rest of the code

# 2. Basic Syntax and Importing Libraries

**Input**

- Getting information from the user from input menu.
- Stored as String.

**Syntax**

```
name = input('Enter your name')
```

# 2. Basic Syntax and Importing Libraries

**Comments**

- Documenting code.
- Used in debugging.
- Won't be executed.

**Syntax**

```
# this is a comment
```

```
…
```

```
This is a multi line
comment
```

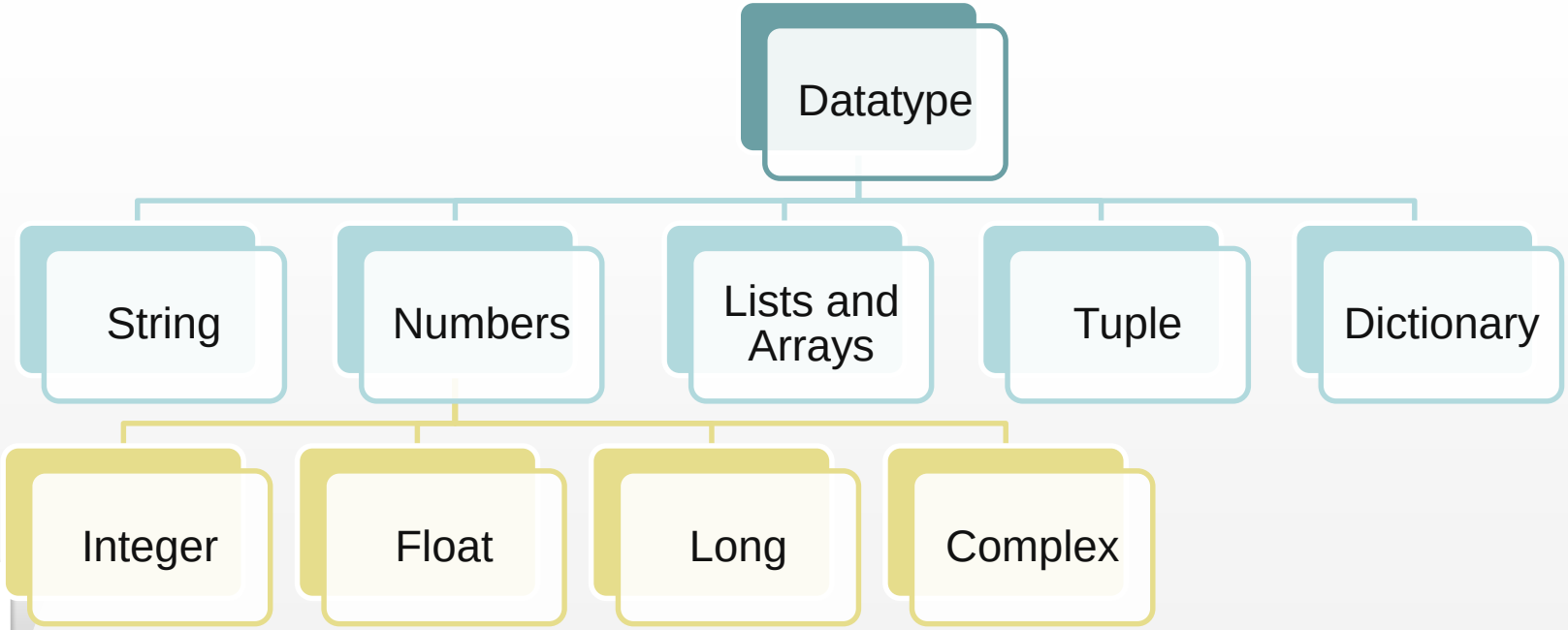
```
…
```

**Visual Studio shortcuts:**

Comment: `Ctrl + K + C`

Uncomment: `Ctrl + K + U`

# 2. Basic Syntax and Importing Libraries



# 2. Basic Syntax and Importing Libraries

**Strings**

- Variables acting as placeholders for value inside the code.
- No keyword in Python (int, String … etc).

**Syntax**

```
first_name = 'Mariham'
last_name = 'Ibrahim'

#concatenation by + or space
Print('Hello' + first_name + ' ' + last_name)
```

As space

# 2. Basic Syntax and Importing Libraries

| Modify Strings | Syntax |
| --- | --- |
| Uppercase | `first_name.upper()` |
| Lowercase | `first_name.lower()` |
| Title | `first_name.title()` |
| Capitalize 1st letter only | `first_name.capitalize()` |
| Count characters | `first_name.count('a')` |
| String Length | `len(first_name)` |
| Repetition | `first_name*2` |

# 2. Basic Syntax and Importing Libraries

| Modify Strings | Syntax | Counting items start at 0 |
| --- | --- | --- |
| Subscript | `first_name[2]` | |
| Slice Range | `first_name[1:4]` | 1 inclusive<br>4 exclusive |
| From Beginning Slice | `first_name[:4]` | |
| Till ending Slice | `first_name[2:]` | |
| Last Letter | `first_name[-1]` | |
| Is Number | `first_name.isdigit()` | |
| Is Capital Letters | `first_name.isupper()` | |

# 2. Basic Syntax and Importing Libraries

**Custom String formatting concatenation**

```
first_name = 'Mariham'
last_name = 'Ibrahim'
Output = 'Hello, ' + first_name + ' ' + last_name
Output = 'Hello, {} {}'.format(first_name,last_name)
Output = 'Hello, {0} {1}'.format(first_name,last_name)
Output = f'Hello, {first_name} {last_name}' #works in
Python 3 only
```

# 2. Basic Syntax and Importing Libraries

## Numbers

- Stored as variables.
- Python supports four different numerical types:

```
int
long
float
complex
```

## Symbols and Operations

| Symbol | Operation |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| ** | Exponent |

# 2. Basic Syntax and Importing Libraries

**Numbers**

- Assign value
- Multiple assignment
- Multiple assignment different datatype

**Syntax**

```
a = 5
a,b,c = 1,2,3
a,b,c = 1,2.5,'Hi'
```

# 2. Basic Syntax and Importing Libraries

**Combining String and Numbers**

- Combine different datatype in the same line, Python will get confused.
- Numeric values are used for math operations and to specify individual row in lists and arrays.

**Syntax**

```
num = '5'
str(5)
int(num) #whole number
float(num) #decimal number
```

# 2. Basic Syntax and Importing Libraries

**List**

- Collection of item of different datatype such as Object, String and numbers defined by [ ] square brackets.
- Zero based index
- Storage order guaranteed
- Can’t change them

**Syntax**

```
names = [‘Mira’, ‘Farida’]
scores = []
scores.append(98)
scores[1]
```

# 2. Basic Syntax and Importing Libraries

| Modify Lists | Syntax |
| --- | --- |
| Subscript | `names[2]` |
| Slice Range | `names[1:4]` |
| From Beginning Slice | `names[:2]` |
| Till ending Slice | `names[1:]` |
| Last Letter | `names[-1]` |
| Sort | `names.sort()` |
| Insert | `names.insert(0,'Maggie')` |
| List Length | `len(names)` |

Counting items start at 0

1 inclusive
4 exclusive

Ascending order
if in descending order
sort(reverse = true)

index

Object

# 2. Basic Syntax and Importing Libraries

**Modify Lists**

- Concatenation
- Repetition

**Syntax**

```
names + names
```

```
names*2
```

# 2. Basic Syntax and Importing Libraries

**Array**

- Collection of item numerical datatype.
- Zero based index
- Storage order not guaranteed
- Must all items be same datatype

**Syntax**

```
from array import array
scores = array('d')
scores.append(98)
```

Double type
If integer 'I'
If float 'f'

# 2. Basic Syntax and Importing Libraries

| Modify Arrays | Syntax |
| --- | --- |
| Subscript | `scores[2]` |
| Slice Range | `scores[1:4]` |
| From Beginning Slice | `scores[:2]` |
| Till ending Slice | `scores[1:]` |
| Last Letter | `scores[-1]` |
| Sort | `scores.sort()` |
| Insert | `scores.insert(0,0.96)` |
| List Length | `len(scores)` |

1 inclusive
4 exclusive

Ascending order
if in descending order
sort(reverse = true)

index Object

# 2. Basic Syntax and Importing Libraries

**Modify Arrays**

- Concatenation
- Repetition

**Syntax**

```
scores + scores
scores*2
```

# 2. Basic Syntax and Importing Libraries

**Tuple**

- Contains items of different datatypes separated by commas and enclosed with () parentheses
- Can’t be updated (read only)

**Syntax**

```
tuple = (‘Mira’, 2.2, ‘Maggie’, 7)
tuple.append(6)
```

Syntax invalid

# 2. Basic Syntax and Importing Libraries

**Dictionary**

- Hash table type
- Key value pairs
- Different datatype
- Enclosed by {} curly braces
- Values assigned and accessed using []
- As stack (last in first out)
- Storage order not guaranteed

**Syntax**

```
dict = {}
dict['one'] = 'This is one' #print(dict['one'])
dict = {'two' : 'This is Two'}
dict[3] = 'This is three'
# print(dict[3])
dict.keys()
dict.values()
```

Key
Value
Key
Value
Index is the key

# 2. Basic Syntax and Importing Libraries

**Import libraries - Dates**

- We often need current date and time when debugging errors and saving data.

**Syntax**

```
from datetime import datetime

current_date = datetime.now() #stored as Date datatype
```

# 2. Basic Syntax and Importing Libraries

**Import libraries - Dates**

- `timedelta`: used to define a period of time (date magic as doing math of dates), used to add or remove days and weeks.

**Syntax**

```
from datetime import datetime, timedelta

current_date = datetime.now()

one_day = timedelta (days = 1)

yesterday = current_date - one_day #stored as Date datatype
```

# 2. Basic Syntax and Importing Libraries

**Import libraries - Dates**

- `timedelta`: used to define a period of time (date magic as doing math of dates), used to add or remove days and weeks.

**Syntax**

```
from datetime import datetime, timedelta

current_date = datetime.now()

one_day = timedelta (days = 1)

tomorrow = current_date + one_day #stored as Date datatype
```

# 2. Basic Syntax and Importing Libraries

**Import libraries - Dates**

- `timedelta`: used to define a period of time (date magic as doing math of dates), used to add or remove days and weeks.

**Syntax**

```
from datetime import
datetime, timedelta

current_date =
datetime.now()

one_week = timedelta
(weeks = 1)

one_week_before =
current_date - one_week
#stored as Date datatype
```

# 2. Basic Syntax and Importing Libraries

## Import libraries - Dates

- Extracting part of date such as day, month, year, hour, minute and second.

## Syntax

```
from datetime import datetime, timedelta
current_date = datetime.now()
day = current_date.day
year = current_date.year
hour = current_date.hour
```

# 2. Basic Syntax and Importing Libraries

## Import libraries - Dates

- `strptime`: convert date from String to date format.

## Syntax

```
from datetime import datetime, timedelta

birthday = input('when is your birthday (dd/mm/yy)?')

birthday_date = datetime.strptime(birthday, '%d/%m/%Y')
```



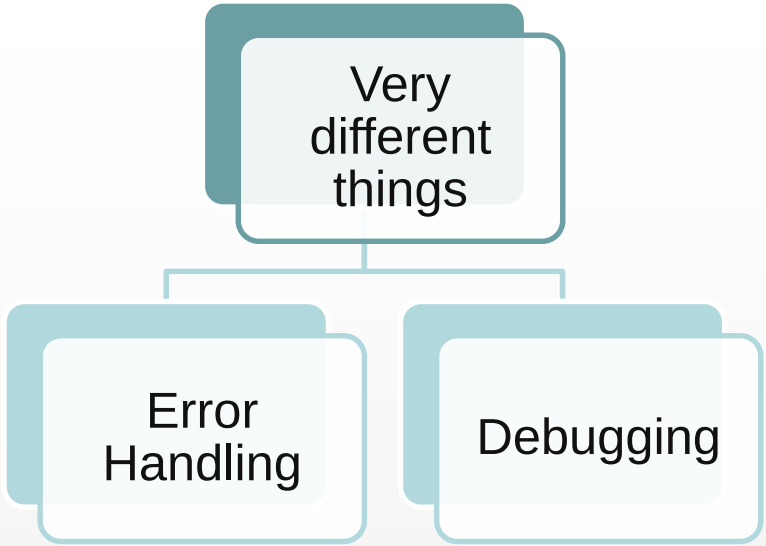
Error handle with exception if someone enters date in wrong format ex: 22-10-2020 or 22/10


# Error Handling

3


“
It’s not a bug – it’s an undocumented feature
Anonymous

# 3. Error Handling



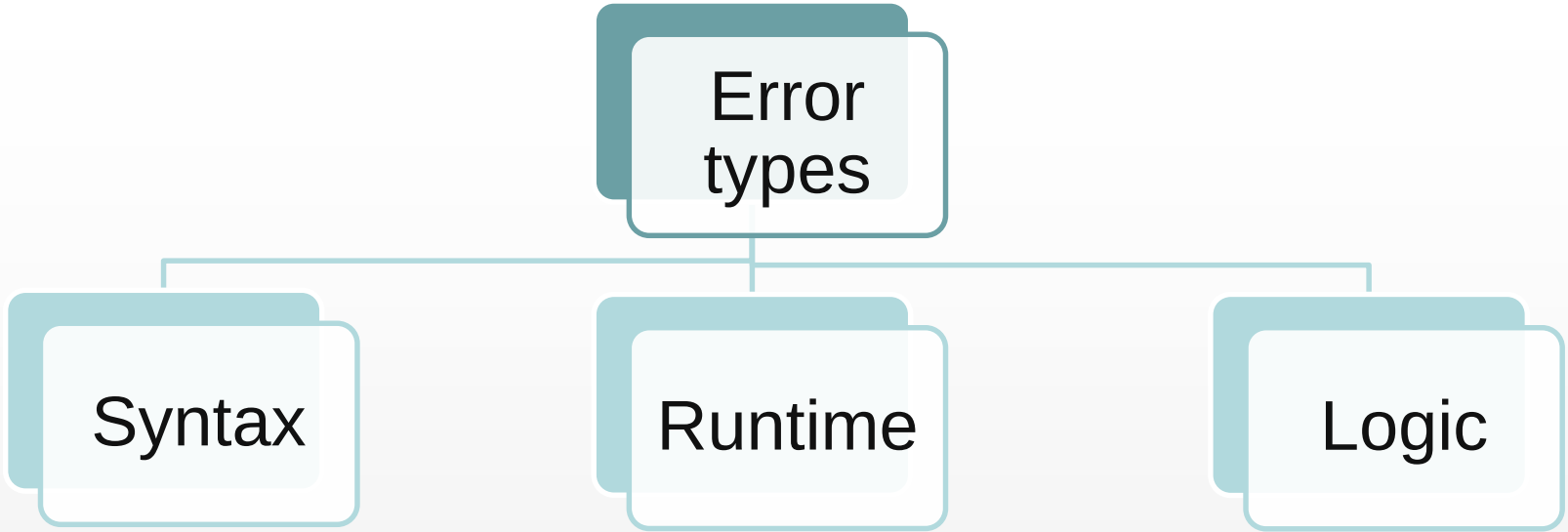

Problem with running code
Unable to predict error

- Permission issues
- Database changes
- Server down

I know there is a problem with my code
Something wrong I done

- Wrong output
- Crashing

# 3. Error Handling




- Code won't run at all
- Easy try and track down
- Error message

```
If x = y:
```

Forget colon

- Code will fail when run
- Decent error
- Error inside code

```
x/y
```

- Code will fail when run
- Compile properly

```
x,y = 10,5
If x <  y:
```

# 3. Error Handling

## Try except else finally

### Syntax

When something went wrong or I specify error

Pass: evaluate this and go on next

Exception name get it from docs or try error by yourself

Identifier

```
try:
    Print(2/0)
except ZeroDivisionError as e:
    print('sorry')
else:
    print(answer)
finally
    print('Print this in all cases')
```

Always run on success or failure


# Handling Conditions and Loops

4

# 4. Handling Conditions and Loops

## If statements

- Conditional logic

| Symbol | Operation |
|---|---|
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |
| == | Equal to |
| != | Not equal to |

## Syntax

```
if price >= 1.00:
    tax = 0.7
else:
    tax = 0
```

Default action

4 spaces indentation means block (suite)
Instead of curly brackets

# 4. Handling Conditions and Loops

**If statements**

- Strings comparisons are case sensitive.

**Syntax**

```
if ‘Mira’.casefold() ==
‘mira’.casefold():
    print(‘true’)
else:
    print(‘false’)
```

# 4. Handling Conditions and Loops

**If statements spacing**

**Syntax**

```
if x == y:
    print('true')
```

```
if x == y:
    State = true
print('true')
```

# 4. Handling Conditions and Loops

**Multiple Conditions using “elif”**

**Syntax**

```
if x == y:
    print('x is equal to y')
elif x == z:
    print('x is equal to z')
else:
    print('x is not equal to y or z')
print('Done')
```

# 4. Handling Conditions and Loops

## Combined Conditions Using “or”

| First Condition | Second Condition | Condition Evaluation |
|---|---|---|
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

## Syntax

```
x = 1
if x == 1 or x == 2:
    print('true')
else:
    print('false')
```

# 4. Handling Conditions and Loops

## Combined Conditions Using “and”

| First Condition | Second Condition | Condition Evaluation |
|---|---|---|
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

## Syntax

```
x,y = 1,2
if x == 1 and y == 2:
    print('true')
else:
    print('false')
```

# 4. Handling Conditions and Loops

**List of possible condition using “in”**

**Syntax**

```
student_name = ‘Mariham’
if student_name in (‘Networks’, ‘Communication’, ‘Electronics’):
   print(‘IET’)
else:
   print(‘Engineering’)
```

# 4. Handling Conditions and Loops

**Nested “ifs”**
**Syntax**

```
gpa = 1.09
honour_grade_max = 0.9
if gpa <= 1.54:
    if honour_grade_max <= 1.69:
        print('Excellent with Honours')
    else:
        print('Excellent only')
else:
    print('Very good or good')
```

# 4. Handling Conditions and Loops

## Nested “ifs” ↔ Combined Conditions using “and” Syntax

```
gpa = 1.09
honour_grade_max = 0.9
if gpa <= 1.54:
    if honour_grade_max <= 1.69:
        print('Excellent with
        Honours')
    else:
        print('Excellent only')
else:
    print('Very good or good')
```

```
gpa = 1.09
honour_grade_max = 0.9
if gpa <= 1.54 and
honour_grade_max <= 1.69:
    print('Excellent with
    Honours')
else:
    print('Very good or good')
```

# 4. Handling Conditions and Loops

## If conditions and Boolean Syntax

```
gpa = 1.09
if gpa <= 1.54:
    excellent = True
else:
    excellent = False
```

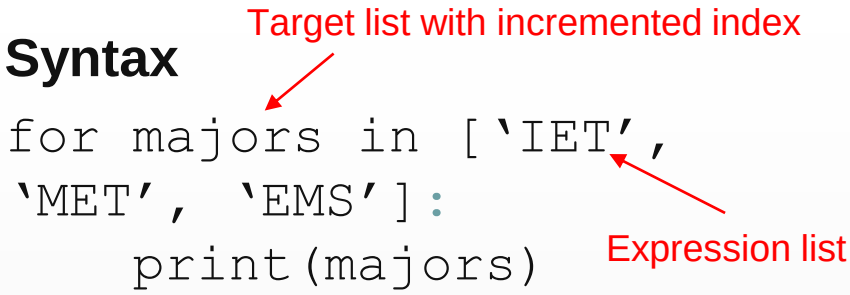
# 4. Handling Conditions and Loops

## Loops

### Unconditional looping using "for"

- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

## Syntax



# 4. Handling Conditions and Loops

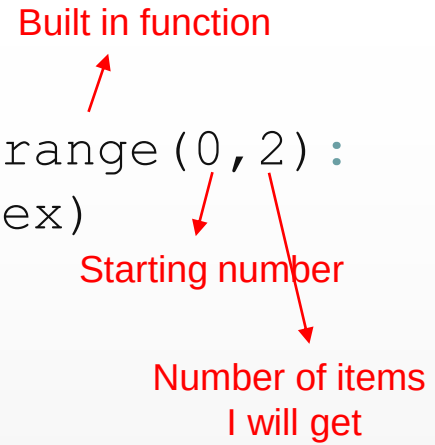
## Loops

### Unconditional looping using “for”

- Looping a number of times using “in range”

## Syntax

```
for index in range(0,2):
    print(index)
```



# 4. Handling Conditions and Loops

## Loops

## Conditional looping using "while"

- Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

## Syntax

```
majors = ['IET', 'MET',
'EMS']
index = 0   #counter
while index < len(majors):
    print(majors[index])
    index = index + 1
```

Specify condition


# Functions and Classes Definition

5

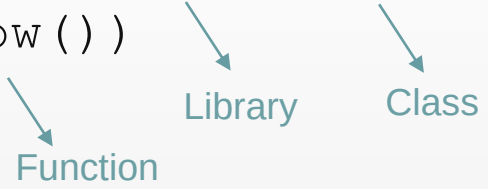
# 5. Functions and Classes Definition

## Functions

- Block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

## Syntax

```
import datetime
def print_time():
    print(datetime.datet
    ime.now())
```




We wrote datetime twice as we didn't specify in import line from datetime (library)

# 5. Functions and Classes Definition

**Functions return a value**

**Syntax**

```
def get_intial(name):
    initial = name[0:1].upper()
    return initial
```

Store in variable assigned to called function

# 5. Functions and Classes Definition

## Functions Calling

### Syntax

```
first_name = ‘mira’
first_name_initial = get_intial(first_name)   #M
```

# 5. Functions and Classes Definition

**Parameterized Functions:** accept multiple parameters

**Syntax**

```
def get_intial(name, force_uppercase):
    if force_uppercase:
        initial = name[0:1].upper()
    else:
        initial = name[0:1]
    return initial
```

Pass the parameters in the same order they are listed in the function declaration

# 5. Functions and Classes Definition

**Parameterized Functions:** accept multiple parameters

**Syntax**

```
def get_intial(name, force_uppercase = True):
    if force_uppercase:
        initial = name[0:1].upper()
    else:
        initial = name[0:1]
    return initial
```

You can specify a default value for a parameter just in case the function is called with only one parameter passed

# 5. Functions and Classes Definition

## Parameterized Functions Calling Positional Notation

### Syntax

```
first_name = 'mira'
force_uppercase = False
first_name_initial = get_initial(first_name,
force_uppercase)
```

Enter parameters in the
**same order** of declaration

# 5. Functions and Classes Definition

## Parameterized Functions Calling Named Notation

### Syntax

```
first_name = 'mira'
first_name_initial = get_initial(force_uppercase
= True, name = first_name)
```

You can specify parameters in any order that makes code more readable

**Given that I know the variable names of passed parameters in function itself**

# 5. Functions and Classes Definition

## Nesting Calling Functions

### Syntax

```
gpa = float(input('What was your gpa?'))
```

# 5. Functions and Classes Definition

## Functions and complex Sort

Allow you to pass in function to call for each list element before it compares items for sorting **using key parameter**

Syntax

```
presenters = [{'Name': 'Mira', 'Age': 26}, [{'Name': 'Farida', 'Age': 29}]
presenters.sort()
```

Sort can automatically handle primitive type and string but will blow up with an error
Sort by Name or Age?

# 5. Functions and Classes Definition

## Functions and complex sort

Define function first then pass key in function

### Syntax

Parameter passed to function specifically

```
def sorter (item):
    return item ['name'] → Return value

presenters = [{'Name': 'Mira', 'Age': 26}, [{'Name':
'Farida', 'Age': 29}]
presenters.sort(key = sorter)
```

# 5. Functions and Classes Definition

## Lambdas

- Single line function that is a clever tool for code cleaning.

**Syntax**

```
presenters = [{'Name': 'Mira', 'Age': 26},
[{'Name': 'Farida', 'Age': 29}]

presenters.sort(key = lambda item: item['Name'])
```

Parameter passed to function specifically

Loop on the items in the list

Return value

# 5. Functions and Classes Definition

## Lambdas

- Sort by length of the name.

**Syntax**

```
presenters = [{'Name': 'Mira', 'Age': 26},
[{'Name': 'Farida', 'Age': 29}]
presenters.sort(key = lambda item: len(item['Name']))
```

# 5. Functions and Classes Definition

## Classes

- Object oriented
- Define data structure and behavior
- Why use classes?
  - Create reusable components
  - Group data and operation together

### Syntax

PascalCasing

this: current instance of object

```
class Presenter():
    def __init__(self,name):
        self.name = name
    def say_Hello(self):
        print('Hello, ' +
        self.name)
```

Constructor To create instances

Field property (setter)

Define the attributes inside constructor on the fly

Method

# 5. Functions and Classes Definition

## Calling Classes

### Syntax

```
presenter = Presenter('Mariham') #defining a class
(new)
Presenter.name = ('Mira')  #updating name
Presenter.say_hello()
```

# 5. Functions and Classes Definition

## Classes Property

### Syntax

```
class Presenter():
    def __init__(self,name):
        self.name = name
    @property
    def name(self):
        return self.name
    @name.setter
    def name(self,value):
        self._name = value
```

Getter
x = presenter.name

Setter
presenter.name = ‘Mira’

# 5. Functions and Classes Definition

## Inheritance

- The transfer of the characteristics of a class to other classes that are derived from it.
- Generalization specialization
- Create an “is a” relationship

# 5. Functions and Classes Definition

## Inheritance

### Syntax

```
class Person(): Parent Class
    def __init__(self,name):
        self.name = name


class Student(Person): Child Class
    def __init__(self,name,school):
        super().__init__((name)) #name call parent's
        constructor
         self.school = school    #school constructor
```

To access parent class

# 5. Functions and Classes Definition

## Inheritance

### Syntax - Parent Class

```
class Person():
    def __init__(self,name):
        self.name = name
    def say_Hello(self):
        print('Hello, ' + self.name)
```

# 5. Functions and Classes Definition

## Inheritance

### Syntax - Child Class

```
class Student(Person):
    def __init__(self,name,school):
        super().__init__((name))
        self.school = school
    def school_name(self):
        print(self.school)

isintance(student,Student)    #True
isintance(student,Person)     #True
issubclass(Student,Person)    #True
```

# 5. Functions and Classes Definition

## Inheritance

### Syntax - Override print function in parent class

```
class Person():
    def __init__(self,name):
        self.name = name
    def say_Hello(self):
        print('Hello, ' + self.name)
    def __str__(self):
        return self.name
```

Without this method it would print as an object

# 5. Functions and Classes Definition

## Inheritance

### Syntax - Override say_hello function in child class

```
class Student(Person):
    def __init__(self,name,school):
        super().__init__((name))
        self.school = school
    def school_name(self):
        print(self.school)
    def say_hello(self):
        super().say_hello()
        print('Name:' + name + \n +'School:' +
       school)
```

# 5. Functions and Classes Definition

## Accessibility in Python

- Everything by default is PUBLIC.
- Single underscore before attribute considered as protected.
  Avoid use it unless you really know what you are doing and need it.
  ```
  _firstname
  ```
- Double underscore before attribute considered as private.
  don’t use it.
  ```
  __firstname
  ```

# Thank You!

## Any questions?

You can find me at:

mariham.Ibrahim@guc.edu.eg

# Credits

- Presentation template by SlidesCarnival
- Documentation:
  https://www.tutorialspoint.com/python/python_classes_objects.htm
- Videos “Microsoft Developer Python for beginners”:
  https://www.youtube.com/playlist?list=PLlrxD0HtieHhS8VzuMCfQD4uJ9yne1mE6
  https://www.youtube.com/playlist?list=PLlrxD0HtieHiXd-nEby-TMCoUNwhbLUnj
- Online compiler: https://www.tutorialspoint.com/execute_python3_online.php