KHWARIZMI
SCIENCE SOCIETY

CODING WITH PYTHON

A COURSE FOR ABSOLUTE BEGINNERS

iDRACK

KHWARIZMI
SCIENCE SOCIETY

iDRACK

https://idrack.org/

https://idrack.org/forum/community/

contact@idrack.org

COURSE OUTLINE

KHWARIZMI SCIENCE SOCIETY

5 WEEKS, SATURDAYS, 2-5 PM

Week 1:

- An introduction to Python
- Installing Python on Windows
- Python Shell

Week 2:

- Saving & Running scripts
- Operators & Variables
- Working with Strings

Week 3:

- Python Collections
- Condition Blocks

Week 4:

- Writing Loops
- Functions in Python

Week 5:

- Introducing Modules
- In-class Assessment

iDRACK

INSTRUCTOR CONTACT

ROOP OMAR

(roop.omar@gmail.com)

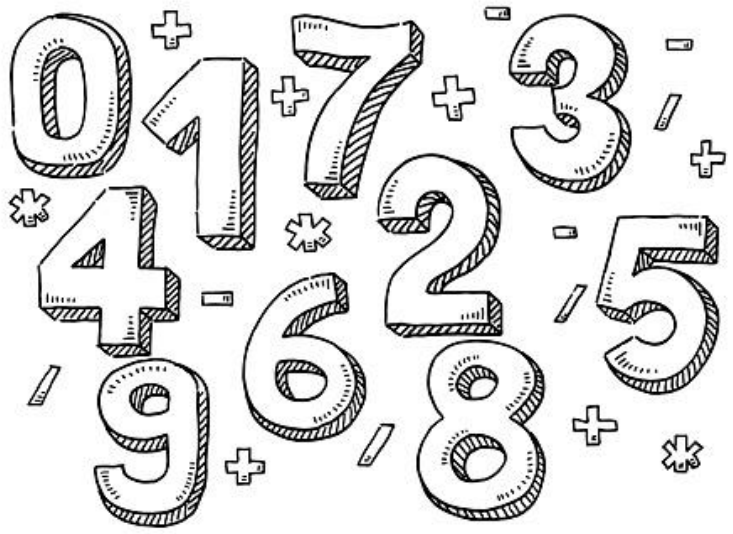
We will try to cater to all queries within class timings, but if you feel there is something you need help with later, or were not able to ask during the session, please feel free to drop me an email, and I will get back to you as soon as I can.

KHWARIZMI
SCIENCE SOCIETY

DATA-TYPES IN PYTHON

iDRACK

PYTHON NUMBERS



- These include:
  - ✓ Integer
  - ✓ Floats
  - ✓ Complex number
- they are represented as int, float and complex.
- **Integers** - can hold a value of any length, the only limitation being the amount of memory available.
- **Float, or "floating point number"** - is only accurate up to 15 decimal places. After that, it rounds the number off.
- **Complex numbers** - are written with a "j" as the imaginary part.

KHWARIZMI SCIENCE SOCIETY

iDRACK

STRINGS IN PYTHON

KHWARIZMI SCIENCE SOCIETY

- A string is simply a series of characters.
- Anything inside quotes is considered a string in Python
- You can use single or double quotes around your strings

```
1.  "This is a string."
2.  'This is also a string.'
```

- This flexibility allows you to use quotes and apostrophes within your strings

LISTS IN PYTHON

- A list is a collection of items in a particular order.
- It can include
  - ✓ the letters of the alphabet
  - ✓ the digits from 0–9
  - ✓ the names of all the people in your family.
- You can put anything you want into a list
- The items in your list don't have to be related.

- The items in a list are separated by commas and enclosed in square braces.

```
assets = ["Computer", "Printer", "TV", "Camera"]

scores = [56, 45.9, 89.5, 70, 32.9, 67.4]

letters = ['k', 'i', 'n', 'd', 's', 'o', 'n']

things = ["chair", 45, 'A', "house"]
```

- Python has a set of built-in methods that you can use on lists.

append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

- In Lists, we can take portions (including the lower but not the upper limit).
- List slicing is the method of splitting a subset of a list,

```
a[start:stop]  # items start through stop-1
a[start:]      # items start through the rest of the array
a[:stop]       # items from the beginning through stop-1
a[:]           # a copy of the whole array
```

- Python has 3 methods for deleting list elements:
  - ✓ *list.remove()*
  - ✓ *list.pop()*
  - ✓ *del* operator

Python List – A Beginners Tutorial to Get Started Quickly

https://www.techbeamers.com/python-list/

- *remove* method takes the particular element to be removed as an argument and deletes the first occurrence of number mentioned in its arguments.
- *pop* and *del* take the index of the element to be removed as an argument.
- *del(a:b)* - deletes all the elements in range starting from index 'a' till 'b'.
- *clear()* - This function is used to erase all the elements of list. List becomes empty.

```
vowels = ['a','e','i','o','u']

vowels.remove('a')

# Result: ['e', 'i', 'o', 'u']
print(vowels)

# Result: 'i'
print(vowels.pop(1))

# Result: ['e', 'o', 'u']
print(vowels)

# Result: 'u'
print(vowels.pop())

# Result: ['e', 'o']
print(vowels)

vowels.clear()

# Result: []
print(vowels)
```

KHWARIZMI SCIENCE SOCIETY

iDRACK

- Python list implements the *sort()* method for ordering its elements in place.
- By default, the function *sort()* performs sorting in the ascending sequence.

```
theList = ['a','e','i','o','u']
theList.sort()
print(theList)
```

```
['a', 'e', 'i', 'o', 'u']
```

```
theList = ['a','e','i','o','u']
theList.sort(reverse=True)
print(theList)
```

```
['u', 'o', 'i', 'e', 'a']
```

- You can use the built-in sorted() function to return a copy of the list with its elements ordered.

```
theList = ['a','e','i','o','u']
newList = sorted(theList)
```

```
theList = ['a','e','i','o','u']
newList = sorted(theList, reverse=True)
```

TUPLES IN PYTHON

KHWARIZMI SCIENCE SOCIETY

- A tuple in python is also a collection of items just like a list.
- Take note of the two key differences:
  - ✓ A tuple is immutable (you can't change the elements once created)
  - ✓ A tuple is created with rounded braces

```
assets = ("Computer", "Printer", "TV", "Camera")

scores = (56, 45.9, 89.5, 70, 32.9, 67.4)

letters = ('k', 'i', 'n', 'd', 's', 'o', 'n')

things = ("chair", 45, 'A', "house")
```

iDRACK

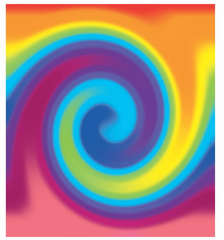
- Python has two built-in methods that you can use on tuples.

index()	Searches the tuple for a specified value and returns the position of where it was found
count()	Returns the number of times a specified value occurs in a tuple

- Unlike lists, the tuple items can not be deleted by using the ***del*** keyword because tuples being immutable.
- To delete an entire tuple, we can use the ***del*** keyword with the tuple name.

```
del tupleA[0]
```

```
del tupleA
```



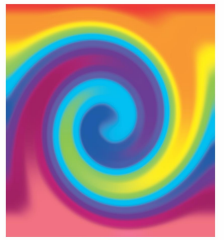

```
# create an empty tuple
py_tuple = ()
print("A blank tuple:", py_tuple)

# create a tuple without using round brackets
py_tuple = 33, 55, 77
print("A tuple set without parenthesis:", py_tuple, "type:", type(py_tuple))

# create a tuple of numbers
py_tuple = (33, 55, 77)
print("A tuple of numbers:", py_tuple)

# create a tuple of mixed numbers
# such as integer, float, imaginary
py_tuple = (33, 3.3, 3+3j)
print("A tuple of mixed numbers:", py_tuple)

# create a tuple of mixed data types
# such as numbers, strings, lists
py_tuple = (33, "33", [3, 3])
print("A tuple of mixed data types:", py_tuple)

# create a tuple of tuples
# i.e. a nested tuple
py_tuple = (('x', 'y', 'z'), ('X', 'Y', 'Z'))
print("A tuple of tuples:", py_tuple)
```

```
# output
A blank tuple: ()
A tuple set without parenthesis: (33, 55, 77) type: <class 'tuple'>
A tuple of numbers: (33, 55, 77)
A tuple of mixed numbers: (33, 3.3, (3+3j))
A tuple of mixed data types: (33, '33', [3, 3])
A tuple of tuples: (('x', 'y', 'z'), ('X', 'Y', 'Z'))
```

```
# Indexing the first element
print("OP(vowel_tuple[0]):", vowel_tuple[0])

# Indexing the last element
print("OP(vowel_tuple[length-1]):", vowel_tuple[len(vowel_tuple) - 1])

# Indexing a non-existent member
# will raise the IndexError
try:
print(vowel_tuple[len(vowel_tuple)+1])
except Exception as ex:
print("OP(vowel_tuple[length+1]) Error:", ex)

# Indexing with a non-integer index
# will raise the TypeError
try:
print(vowel_tuple[0.0])
except Exception as ex:
print("OP(vowel_tuple[0.0]) Error:", ex)

# Indexing in a tuple of tuples
t_o_t = (('jan', 'feb', 'mar'), ('sun', 'mon', 'wed'))

# Accessing elements from the first sub tuple
print("OP(t_o_t[0][2]):", t_o_t[0][2])

# Accessing elements from the second sub tuple
print("OP(t_o_t[1][2]):", t_o_t[1][2])
```

```
# output
The tuple: ('a', 'e', 'i', 'o', 'u') Length: 5
OP(vowel_tuple[0]): a
OP(vowel_tuple[length-1]): u
OP(vowel_tuple[length+1]) Error: tuple index out of range
OP(vowel_tuple[0.0]) Error: tuple indices must be integers or slices, not float
OP(t_o_t[0][2]): mar
OP(t_o_t[1][2]): wed
```

```
>>> weekdays = ('mon', 'tue', 'wed' ,'thu', 'fri', 'sat', 'sun')
>>> weekdays
('mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun')
# accessing elements leaving the first one
>>> weekdays[1:]
('tue', 'wed', 'thu', 'fri', 'sat', 'sun')
# accessing elements between the first and fifth positions
# excluding the ones at the first and fifth position
>>> weekdays[1:5]
('tue', 'wed', 'thu', 'fri')
# accessing elements after the fifth position
>>> weekdays[5:]
('sat', 'sun')
# accessing the first five elements
>>> weekdays[:5]
('mon', 'tue', 'wed', 'thu', 'fri')
# accessing elements that appears after
# counting five from the rear end
>>> weekdays[:-5]
('mon', 'tue')
# accessing five elements from the rear
>>> weekdays[-5:]
('wed', 'thu', 'fri', 'sat', 'sun')
# accessing elements from the start to end
>>> weekdays[:]
('mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun')
```



KHWARIZMI
SCIENCE SOCIETY

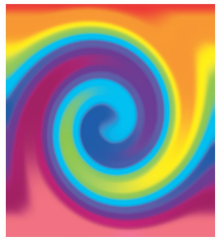
iDRACK

SETS IN PYTHON

- Sets are also like lists
- A set is a collection that is unordered and unindexed.
- The elements don't have a specific order, and their positions can be inconsistent.
- Each item is unique in a set and, therefore, can't have duplicates.
- The elements are immutable and hence, can't accept changes once added.
- A set is itself mutable and allows the addition or deletion of items.
- In Python, sets are written with curly brackets.

- Python has a set of built-in methods that you can use on sets.

add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Removes the items in this set that are also included in another, specified set



intersection()	Returns a set containing the difference between two or more sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have an intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	() Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

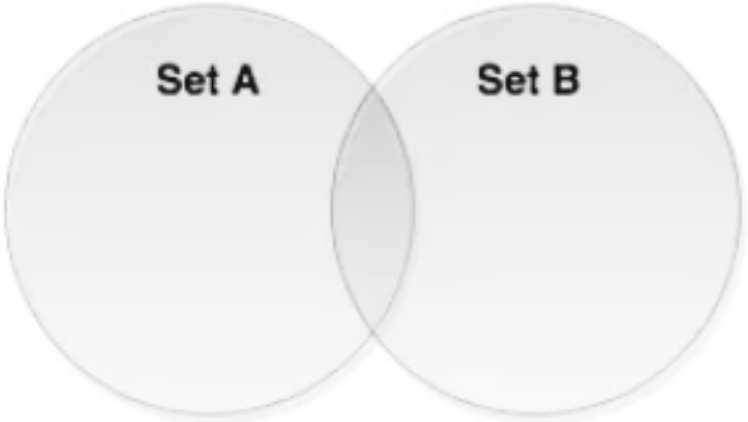
- Union of setA and setB is a new set combining all the elements from both the sets.
- The "|" operator is used to perform the union operation on the sets.
- You can also accomplish similar results using the *union()* method.

```
# We'll use the setA and setB for our illustration
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}
```

```
print("Initial setA:", setA, "size:", len(setA))
print("Initial setB:", setB, "size:", len(setB))

print("(setA | setB):", setA | setB, "size:", len(setA | setB))
```

```
print("setA.union(setB):", setA.union(setB), "size:", len(setA.union(setB)))
print("setB.union(setA):", setB.union(setA), "size:", len(setB.union(setA)))
```

```
# output
Initial setA: {'u', 'i', 'g', 'o', 'e', 'h', 'a'} size: 7
Initial setB: {'u', 'z', 'b', 'o', 'e', 'a', 't'} size: 7
(setA | setB): {'h', 'u', 'z', 'b', 't', 'g', 'o', 'e', 'i', 'a'} size: 10
```

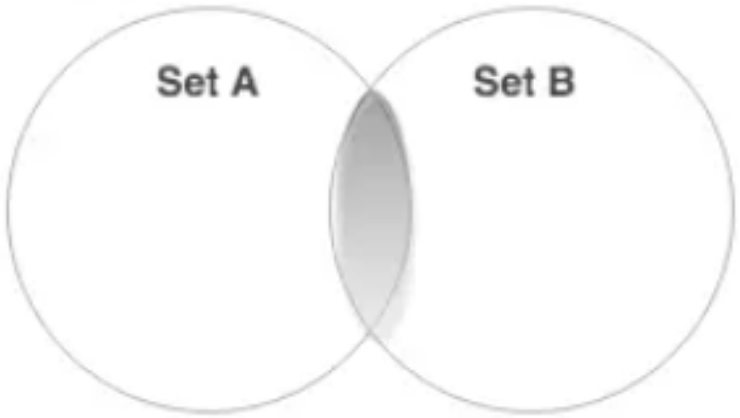


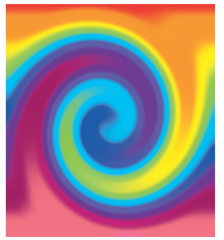
- The intersection of setA and setB will produce a set comprising common elements in both the sets.
- We use Python's "&" operator to perform this operation.
- Alternatively, you can call the *intersection()* method to perform this operation.

```
# Python intersection example using the & operator
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}

print("Initial setA:", setA, "size:", len(setA))
print("Initial setB:", setB, "size:", len(setB))

print("(setA & setB):", setA & setB, "size:", len(setA & setB))
```

```
intersectAB = setA.intersection(setB)
print("setA.intersection(setB):", intersectAB, "size:", len(intersectAB))
intersectBA = setB.intersection(setA)
print("setB.intersection(setA):", intersectBA, "size:", len(intersectBA))
```

```
# output
Initial setA: {'e', 'o', 'h', 'a', 'g', 'u', 'i'} size: 7
Initial setB: {'b', 'e', 't', 'o', 'z', 'a', 'u'} size: 7
(setA & setB): {'o', 'a', 'u', 'e'} size: 4
```





- When you perform the difference operation on two Sets, i.e., <setA – setB>, the resultant will be a set of elements that exist in the left but not in the right object.
- We use the minus (-) operator to carry out this operation.
- The same set of operations can be done using the *difference()* method.

```
# Python set's difference operation
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}

diffAB = setA - setB
print("diffAB:", diffAB, "size:", len(diffAB))
diffBA = setB - setA
print("diffBA:", diffBA, "size:", len(diffBA))
```

```
diffAB = setA.difference(setB)
print("diffAB:", diffAB, "size:", len(diffAB))
diffBA = setB.difference(setA)
print("diffBA:", diffBA, "size:", len(diffBA))
```

```
# output
diffAB: {'i', 'g', 'h'} size: 3
diffBA: {'z', 'b', 't'} size: 3
```

DICTIONARIES IN PYTHON

KHWARIZMI SCIENCE SOCIETY

- A dictionary in Python is a collection of **key-value pairs**.
- Each key is connected to a value
- You can use a key to access the value associated with that key.
- A key's value can be a number, a string, a list, or even another dictionary.
- In Python, dictionaries are written with curly brackets.

```
fruits = {'value':tomato,'key':red}
```

iDRACK

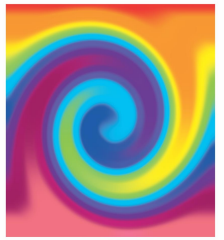
- ✓ keys in a dictionary must be unique (no two same keys)
- ✓ keys are immutable
- ✓ keys and values can be of any data types
- ✓ the *keys()* function returns list of keys in a dictionary
- ✓ the *values()* function returns list of values in dictionary

```
months = {
     "Jan": "January",
     "Feb": "Febraury",
     "Mar": "March",
     "Apr": "April",
     "May": "May",
     "Jun": "June",
     "Jul": "July",
     "Aug": "August",
     "Sep": "September"
}

weekdays = {
     1: "Monday",
     2: "Tuesday",
     3: "Wednesday",
     4: "Thursday",
     5: "Friday",
     6: "Saturday",
     7: "Sunday"
}
```

- Python has a set of built-in methods that you can use on dictionaries.

clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
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
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

POINTS TO REMEMBER



- The **list** is a collection that is ordered and changeable. Allows duplicate members.
- A **tuple** is a collection that is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection that is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection that is unordered, changeable and indexed. No duplicate members.


```python
# Python3 program for explaining
# use of list, tuple, set and
# dictonary

# Lists
l = []

# Adding Element into list
l.append(5)
l.append(10)
print("Adding 5 and 10 in list", l)

# Popping Elements from list
l.pop()
print("Popped one element from list", l)
print()

# Set
s = set()

# Adding element into set
s.add(5)
s.add(10)
print("Adding 5 and 10 in set", s)

# Removing element from set
s.remove(5)
print("Removing 5 from set", s)
print()

# Tuple
t = tuple(l)

# Tuples are immutable
print("Tuple", t)
print()

# Dictonary
d = {}

# Adding the key value pair
d[5] = "Five"
d[10] = "Ten"
print("Dictonary", d)

# Removing key-value pair
del d[10]
print("Dictonary", d)
```

Output:

```
Adding 5 and 10 in list [5, 10]
Popped one element from list [5]

Adding 5 and 10 in set {10, 5}
Removing 5 from set {10}

Tuple (5, )

Dictonary {10: 'Ten', 5: 'Five'}
Dictonary {5: 'Five'}
```

KHWARIZMI
SCIENCE SOCIETY

CONDITION BLOCKS IN PYTHON

iDRACK

WHAT ARE CONDITIONAL STATEMENTS?

KHWARIZMI SCIENCE SOCIETY

- When you're writing a program, you may want a block of code to run only when a certain condition is met.
- That's where conditional statements come in.
- They allow you to control the flow of your program more effectively.
- These are also called condition tests.

  - ✓ *if*
  - ✓ *else*
  - ✓ *elif*
  - ✓ Nested *if*

iDRACK

KHWARIZMI
SCIENCE SOCIETY

IF, ELSE AND ELIF STATEMENTS

iDRACK

"IF" CONDITION

KHWARIZMI SCIENCE SOCIETY

- A Python *if* statement evaluates whether a condition is true or false.
- The statement will execute a block of code if a specified condition is true.
- Otherwise, the block of code within the statement is skipped and not executed.
- *if* condition can be used on simple mathematical conditions such as:

  - ✓ Equal (=)
  - ✓ Not Equal (! =)
  - ✓ Less than (<)
  - ✓ Less than or equal to (<=)
  - ✓ Greater than (>)
  - ✓ Greater than or equal to (>=).

- Conditions may be combined using the keywords **OR** and **AND**.

iDRACK

"IF" CONDITION

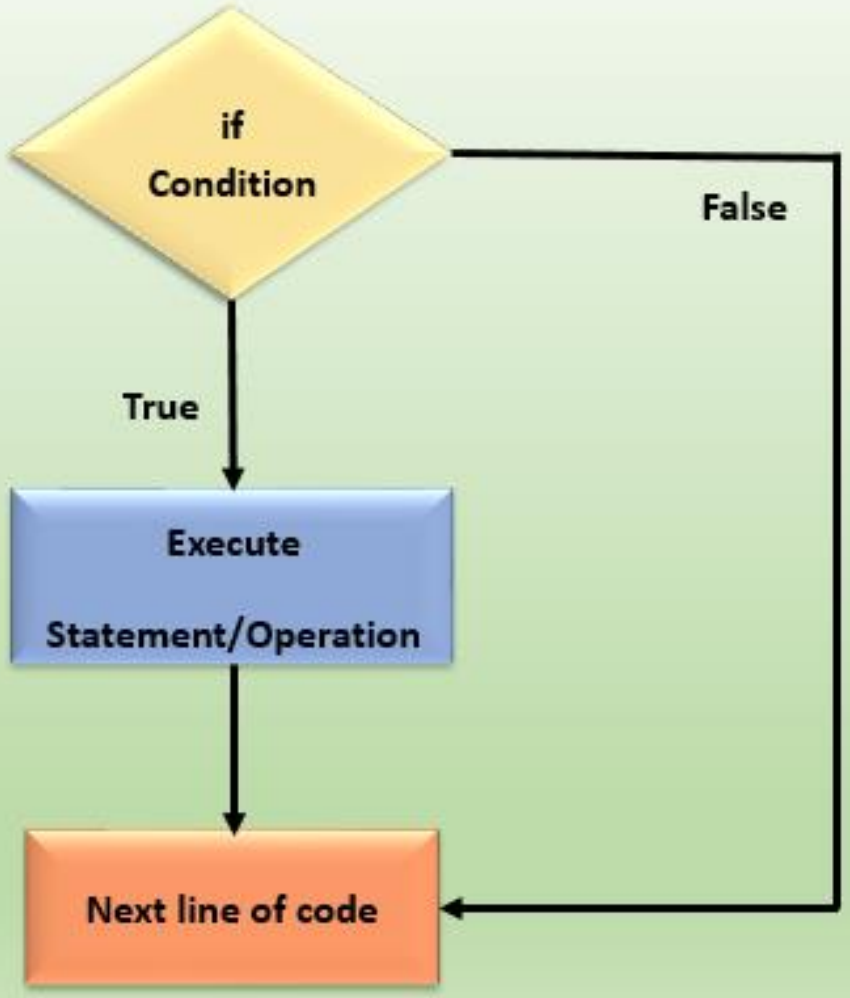
KHWARIZMI SCIENCE SOCIETY

- Python is case sensitive too so *if* should be in lower case.
- Python is sensitive to indentation.
- After the *if* condition, the next line of code is spaced four spaces apart from the start of the statement.
- Any set of instructions or condition that belongs to the same block of code should be indented.

```
if <condition>:
    <statement>
```

iDRACK

If Statement in Python



www.educba.com

"IF" CONDITION

KHWARIZMI SCIENCE SOCIETY

- An *if* statement doesn't need to have a single statement
- It can have a block.
- A block is more than one statement.

```
x = 4
if x < 5:
    print("x is smaller than five")
    print("this means it's not equal to five either")
    print("x is an integer")
```

iDRACK

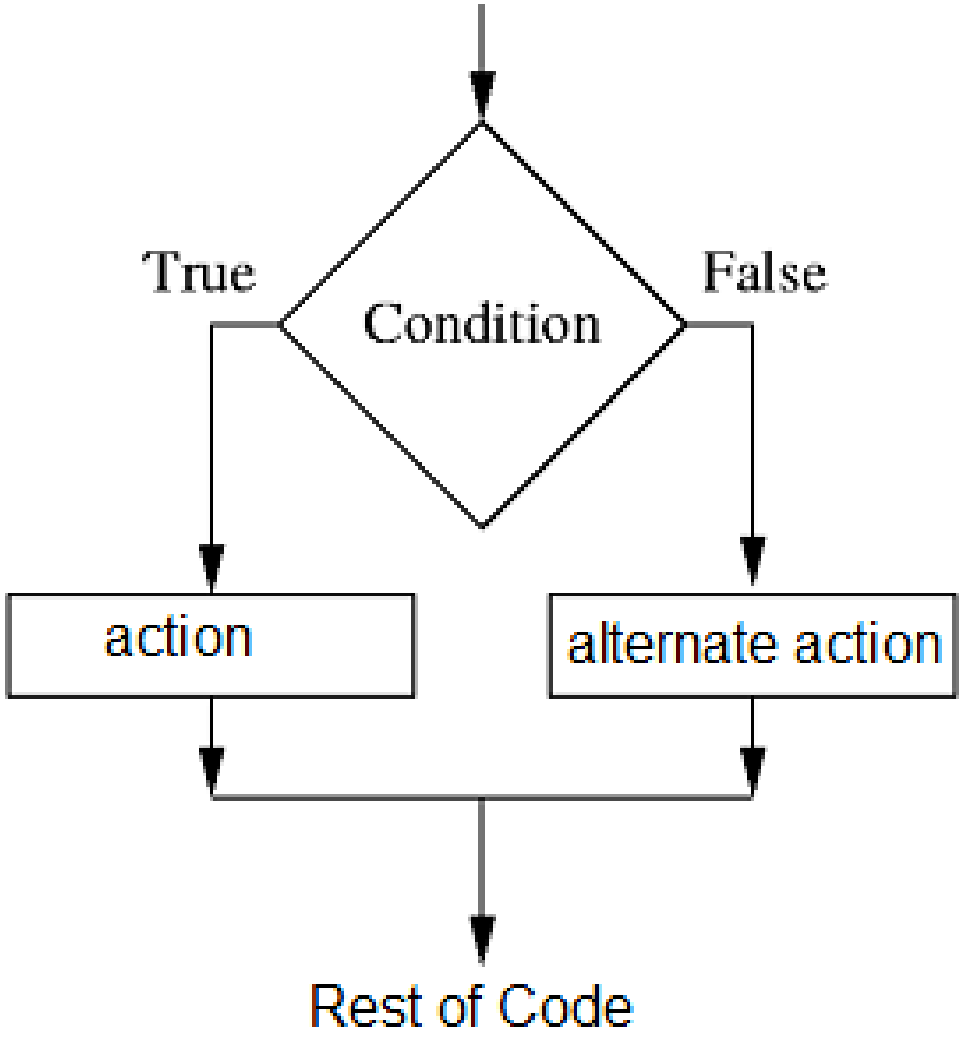
BLOCKS

- All programming languages can create blocks
- Python has a unique way of doing it.
- A block is defined only by its indention.

```
if <condition>:
    <statement>
    <statement>
    <statement>

<statement>  # not in block
```

KHWARIZMI
SCIENCE SOCIETY

iDRACK



"ELSE" CONDITION

KHWARIZMI SCIENCE SOCIETY

- An *else* statement can be combined with an *if* statement.
- An *else* statement contains the block of code that executes if the conditional expression in the *if* statement resolves to 0 or a FALSE value.
- The *else* statement is an optional statement and there could be at most only one *else* statement following *if*.

```
if expression:
   statement(s)
else:
   statement(s)
```

iDRACK

"ELSE" CONDITION

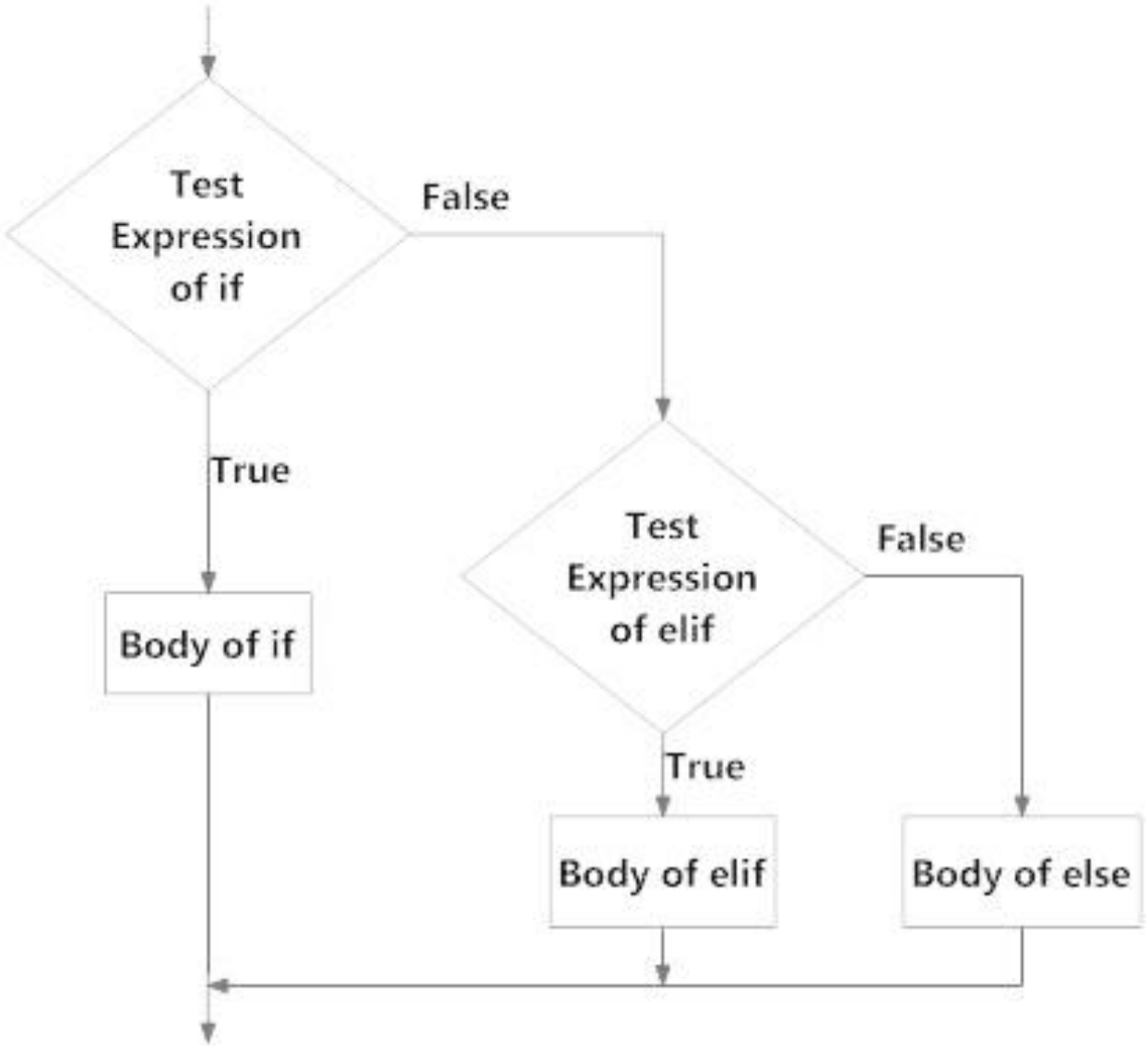
KHWARIZMI
SCIENCE SOCIETY

```
gender = input("Gender? ")
if gender == "male" or gender == "Male":
    print("Your cat is male")
else:
    print("Your cat is female")

age = int(input("Age of your cat? "))
if age < 5:
    print("Your cat is young.")
else:
    print("Your cat is adult.")
```

iDRACK

"ELIF" CONDITION

- The *elif* is short for *else if*.
- It allows us to check for multiple expressions.
- If the condition for *if* is False, it checks the condition of the next *elif* block and so on.
- If all the conditions are False, the body of *else* is executed.
- Only one block among the several *if...elif...else* blocks is executed according to the condition.
- The *if* block can have only one *else* block. But it can have multiple *elif* blocks.




```
>>> x = 3
>>> if x == 2:
...     print('two')
... elif x == 3:
...     print('three')
... elif x == 4:
...     print('four')
... else:
...     print('something else')
...
three
>>>
```

KHWARIZMI SCIENCE SOCIETY

iDRACK

KHWARIZMI
SCIENCE SOCIETY

COMBINING CONDITIONS

iDRACK

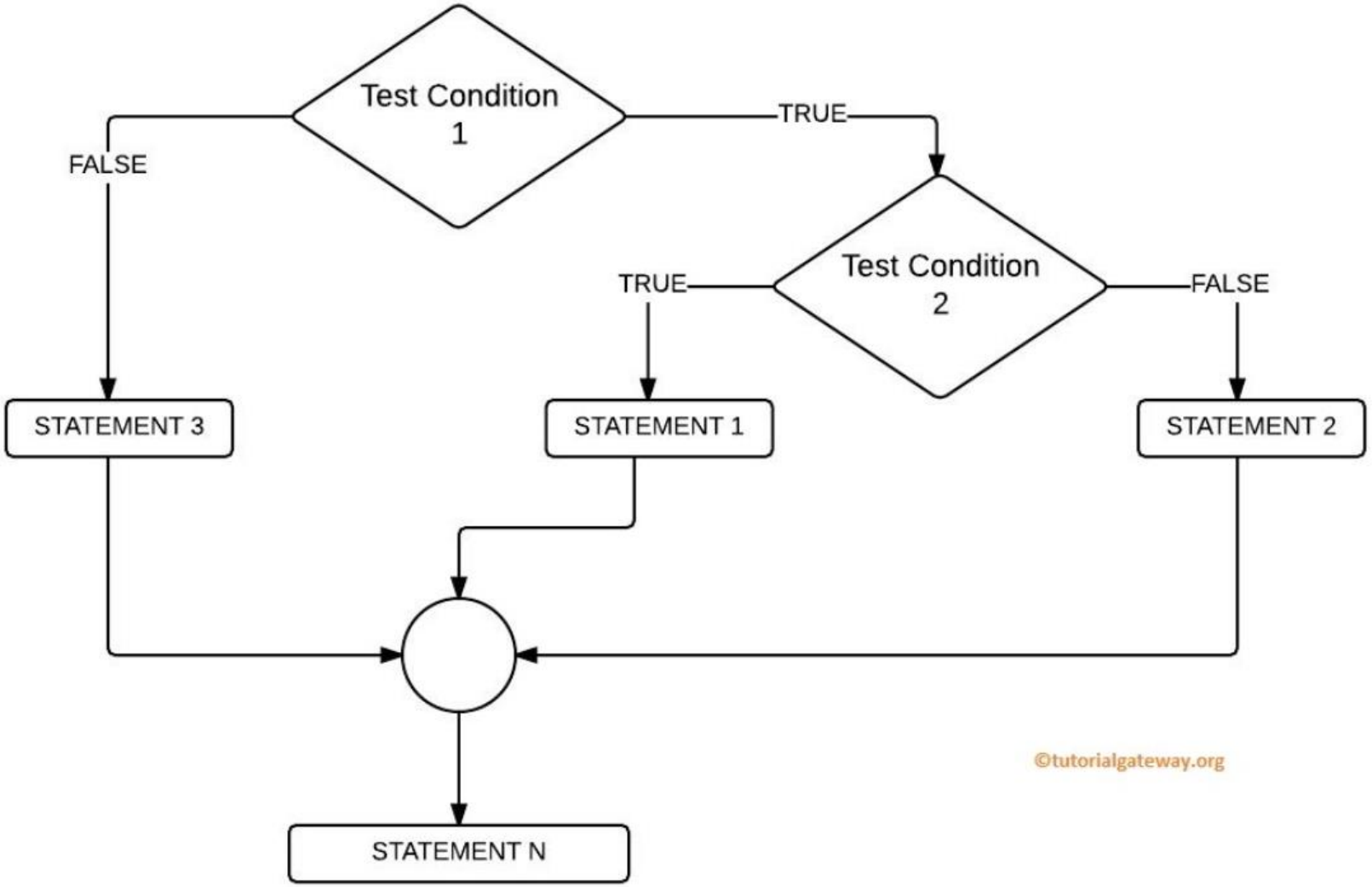
PYTHON NESTED *IF* STATEMENTS

KHWARIZMI SCIENCE SOCIETY

- We can have a *if...elif...else* statement inside another *if...elif...else* statement.
- This is called nesting in computer programming.
- Any number of these statements can be nested inside one another.
- Indentation is the only way to figure out the level of nesting.
- They can get confusing, so they must be avoided unless necessary.

```
if ( test condition 1):
    # If test condition 1 is TRUE, then it checks for test condition 2
    if ( test condition 2):
         # If test condition 2 is TRUE, then these true statements executed
         Test condition 2 True statements
    else:
         # If test condition 2 is FALSE, then these false statements executed
         Test condition 2 False statements
else:
    # If test condition 1 is FALSE, then these statements executed
    Test condition 1 False statements
```

iDRACK

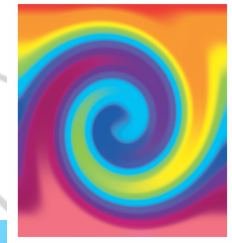


PYTHON NESTED *IF* STATEMENTS

KHWARIZMI
SCIENCE SOCIETY

```
# Example for Python Nested If Statement

age = int(input(" Please Enter Your Age Here:  "))
if age < 18:
    print(" You are Minor ")
    print(" You are not Eligible to Work ")
else:
    if age >= 18 and age <= 60:
        print(" You are Eligible to Work ")
        print(" Please fill in your details and apply")
    else:
        print(" You are too old to work as per the Government rules")
        print(" Please Collect your pension!")
```

iDRACK



Some Exercises

1. Store the names of a few of your friends into a list called ‘names’. Print a message to each one of them, personalized with their name.
2. Make a list of atleast five places in the world you would like to visit. Make sure it is NOT in alphabetical order. Print your list in the original order. Then sort it in ascending and descending orders and print the output. Store the sorted lists in a different variable.
3. Make a program that asks the number between 1 and 10. If the number is out of range the program should display "invalid number".

Take a screenshot of your code and output for record. We will be reviewing this in the next session

KHWARIZMI
SCIENCE SOCIETY

4. Write an if-elif-else program that determines a person's stage of life.

   Set a value for the variable age, and then:

- ✓ If the person is less tha 2 years old, print that the person is a baby
- ✓ If the person is atleast 2 but less than 4, print that the person is a toddler.
- ✓ If the person is atleast 4 but less than 13, print that the person is a kid.
- ✓ If the person is atleast 13 but less than 20, print that the person is a teenager.
- ✓ If the person is atleast 20 but less than 65, print that the person is an adult.
- ✓ If the person is 65 or older, print that the person is an elder.

Take a screenshot of your code and output for record. We will be reviewing this in the next session

iDRACK