Chapter-4 Python conditional and iterative statements

4.1 if statement, if..elif statement, if..elif…else statements, nested if
4.2 Iterative statements :
4.2.1 while loop, nested while loop, break , continue statements.
4.2.2 for loop, range, break, continue, pass and Else with for loop, nested for loop.
4.3 List : creating list, indexing, accessing list members, range in list, List methods (append, clear, copy, count, index, insert, pop, remove, reverse, sort).

4.1 Python If statements

- If statements is used to check condition.
- It executes the further statements based on the result of the given condition
- Types of If Condition:
    1. if statement
    2. if..else statement
    3. if.. elif statement
    4. if.. elif.. else statement
    5. nested if statement
- Python supports the usual logical conditions from mathematics: Eg: Equals: a == b, Not Equals: a != b, Less than: a < b, Less than or equal to: a <= b, Greater than: a > b, Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

1. if statement

- An if statement is written by using the if keyword.

Syntax:	**Example:**	**Output:**

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

Indentation in if statements

- 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.
- All If statement, without indentation (will raise an error):

2. if.. else statement

- if..else statement is used to execute the statements if the given condition is true or false.

Syntax:	**Example**:	**Output**:

if condition:
  statements
else:
  statements
	a = 200
	b = 33
	if b > a:
	  print("b is greater than a")
	else:
	  print("b is not greater than a")
		b is not greater than a

3. if.. elif statement

- The if..elif is use to check multiple conditions and execute accordingly.

- The elif keyword is pythons way of saying "if the previous conditions were not true, then try next condition".

Syntax:	Example	Output

if condition :
  statements
elif condition :
  statements
	a = 33
	b = 33
	if b > a:
	  print("b is greater than a")
	elif a == b:
	  print("a and b are equal")
		a and b are equal

4. **if..elif...else statement**
   - The else keyword executes statements which isn't executed by the preceding conditions.

Syntax:	Example:	Output:

if condition:
  statements
elif condition:
  statements
else:
  statements
	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")
		a is greater than b

5. **Nested if**
   - Using if statements inside if statements, this is called nested if statements.

Syntax:	Example:	Output:

if condition:
  statements
  if condition:
  statement
  else:
  statement
	x = 41
	
	if x > 10:
	  print("Above ten,")
	  if x > 20:
	  print("and also above 20!")
	  else:
	  print("but not above 20.")
		Above ten,
		and also above 20!

4.2 Iterative Statements

- Iterative Statements (Loops) are sequence of statements executed till condition is true.
- Python has two Iterative Statements:
  1. while loops
  2. for loops

1. while loop

- while loop executes a set of statements until a condition is true.
- First of all, the given condition is checked, if it is true then statements are executed.

- When the condition becomes false then the given statements are not executed but the statement outside the body of while statement is executed.
- **Note**: remember to increment i, or else the loop will continue forever.
- The while loop requires index variable to be initialized, in this example we need to define an indexing variable, i, which we set to 1.

Syntax:	**Example**:	**Output**:

while condition:
  statements
	i = 1
	while i < 6:
	  print(i)
	  i += 1
		1
		2
		3
		4
		5

Nested while loop

- A nested while loop is a while loop inside a while loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":

Syntax:	Example:	Output:

while condition:
  statements
  while condition:
  statements
	i = 1
	j = 5
	while i < 4:
	  while j < 8:
	  print(i, " ", j)
	  j = j + 1
	  i = i + 1
		1 5
		2 6
		3 7

The break Statement

- An early exit from loop can be done by using break statement.
- When the break statement is executed within loop then it immediately exited from the loop and executes next statement.
- When nested loop is there then break statement exit only from the current loop.

Example:	Output:

i = 1
while i < 6:
  print(i)
  if i == 3:
  break
  i += 1
	1
	2
	3

The Continue Statements

- continue statement is used to skip a part of the loop once under certain con dition.
- It causes the loop to be continued with the next iteration after skipping any statements in between.

Example:	Output:

i = 0
while i < 6:
  i += 1
  if i == 3:
	1
	2
	4
	5

continue print(i)	6 Note: Here 3 is skipped and rest of numbers are continued.

2. 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).
- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.
- **The for loop does not require an indexing variable to initialized before and nor to increment that variable in the loop, for loop does this task automatically.**

Syntax:	Example:	Output:

for iterating_var in sequence:
  statements(s)
 	
#sequence can be array, dictionary, list etc..
	for l in 'Python': # First Example
	  print('Letter :' + l)
	
	fruits = ['banana', 'apple', 'mango']
	
	for f in fruits: # Second Example
	  print('Current Fruit: ' + f)
		Letter :P
		Letter :y
		Letter :t
		Letter :h
		Letter :o
		Letter :n
		Current Fruit: banana
		Current Fruit: apple
		Current Fruit: mango

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.

Sr No	Examples	Outputs

1	for x in range(4): # range = 0 to 3
	  print(x)
		0
		1
		2
		3

2	# range start from 2 and go till 5 for x in range(2, 6): print(x)	2 3 4 5
3	# range start from 2 and go till 10 increment by 3 for x in range(2, 10, 3): print(x)	2 5 8

- The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6)

Else in For Loop

- The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

Example:	**Output**:

for x in range(4):
  print(x)
else:
  print("Finally finished!")
	0
	1
	2
	3
	Finally finished!

Note: The else block will NOT be executed if the loop is stopped by a break statement.

Example:	**Output:**

for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
	0
	1
	2

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":

Example: Print each adjective for every fruit:	**Output**:

adj = ["red", "big"]
fruits = ["apple", "banana","cherry"]
 	
for x in adj:
  for y in fruits:
  print(x, y)
	red apple
	red banana
	red cherry
	big apple
	big banana
	big cherry

The pass Statement

- for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

Example:	**Output**:
for x in [0, 1, 2]: pass	# having an empty for loop like this, would raise an error without the pass statement

4.3 List

- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets i.e. [ ].

Syntax:

variable_name = [list_Item1, list_Item2,……..]

List Items

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.

Example:	**Output**:
thislist = ["apple", "banana", "cherry"] print(thislist)	['apple', 'banana', 'cherry']

Accessing List Members:

- List items are indexed and to access the elements use the index number:

Example Print the second item of the list: thislist = ["apple", "banana", "cherry"] print(thislist[1])	**Output**: banana

Note: The first item has index **0.**

Negative Indexing

- Negative indexing means start from the end
- -1 refers to the last item, -2 refers to the second last item etc.

Example:	**Output**:
thislist = ["apple", "banana", "cherry"] print(thislist[-1])	cherry

Range of Indexes

- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new list with the specified items.

Example:	**Output:**
Return the third, fourth, and fifth item: thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] print(thislist[2:5])	**['cherry', 'orange', 'kiwi']**

Note: The search will start at index 2 (included) and end at index 5 (not included).

By leaving out the start value, the range will start at the first item:

Example	**Output:**
This example returns the items from the beginning to, but NOT including, "kiwi": thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] print(thislist[:4])	['apple', 'banana', 'cherry', 'orange']

By leaving out the end value, the range will go on to the end of the list:

Example	**Output:**
This example returns the items from "cherry" to the end: thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] print(thislist[2:])	['cherry', 'orange', 'kiwi', 'melon', 'mango']

Range of Negative Indexes

- Specify negative indexes if you want to start the search from the end of the list:

Example	**Output:**
This example returns the items from "orange" (-4) to, but NOT including "mango" (-1): thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] print(thislist[-4:-1])	['orange', 'kiwi', 'melon']

Check if Item Exists

- To determine if a specified item is present in a list use the in keyword:

Example	**Output:**
Check if "apple" is present in the list thislist = ["apple", "banana", "cherry"] if "apple" in thislist: print("Yes, 'apple' is in the fruits list")	Yes, 'apple' is in the fruits list

List Methods:

Sr No	Method	Explanation	Example for list fruits: fruits = ['apple', 'banana', 'cherry']	Output
1	append	Adds an element at the end of the list	fruits.append("orange") print(fruits)	['apple', 'banana', 'cherry', 'orange']
2	clear	Removes all the elements from the list	fruits.clear() print(fruits)	[]
3	copy	Returns a copy of the list	x = fruits.copy() print(x)	['apple', 'banana', 'cherry', 'orange']
4	count	Returns the number of elements with the specified value	x = fruits.count("cherry") print(x)	1
5	index	Returns the index of the first element with the specified value	x = fruits.index("cherry") print(x)	2
6	insert	Adds an element at the specified position	fruits.insert(1, "orange") print(fruits)	['apple', 'orange', 'banana', 'cherry']
7	pop	Removes the element at the specified position	fruits.pop(1) print(fruits)	['apple', 'banana', 'cherry']
8	remove	Removes the item with the specified value	fruits.remove("banana") print(fruits)	['apple', 'cherry']
9	reverse	Reverses the order of the list	fruits.reverse() print(fruits)	[‘cherry’, ‘apple’]
10	sort	Sorts the list	cars = ['Ford', 'BMW', 'Volvo'] cars.sort() print(cars)	['BMW', 'Ford', 'Volvo']