BREAK AND CONTINUE STATEMENTS IN PYTHON

Python provides two important statements to control the flow of loops:

1. break Statement

- The `break` statement **terminates the loop immediately**, even if the loop condition is still true.
- It is generally used when we want to **exit a loop early**, based on a condition.

Example:

```
for i in range(1, 10):
    if i == 5:
        break
    print(i)
```

Output:

```
1
2
3
4
```

⇥ When `i` becomes 5, the `break` statement is executed, and the loop stops.

---

2. continue Statement

- The `continue` statement **skips the current iteration** and moves to the **next iteration** of the loop.
- It is used when we want to **ignore some values or conditions** but still continue the loop.

Example:

```
for i in range(1, 6):

    if i == 3:

        continue

    print(i)
```

Output:

```
1
2
4
5
```

WHAT IS A FUNCTION IN PYTHON?

A **function** is a reusable block of code that performs a specific task. It helps make the code modular and reduces repetition.

Types of Functions

1. **Built-in Functions** – Already provided by Python (e.g. `print()`, `len()`, `range()`)
2. **User-defined Functions** – Created by the user to perform custom tasks

---

Defining a Function

```
def function_name(parameters):

    # code block

    return result
```

✅ `def`: Keyword to define a function
✅ `parameters`: Optional values the function can receive
✅ `return`: Sends a value back to where the function was called

Example:

```
def greet(name):

    return "Hello, " + name

print(greet("Amit"))

# Output: Hello, Amit
```

Calling a Function

You "call" a function by using its name followed by parentheses:

```
function_name(arguments)
```

Parameters vs Arguments

- **Parameter**: Variable in the function definition
- **Argument**: Actual value passed during the function call

Function with Default Parameters

```
def greet(name="Guest"):

    return "Hello, " + name
```

```
print(greet())          # Output: Hello, Guest

print(greet("Riya"))    # Output: Hello, Riya
```

Return Statement

Used to send the output from a function:

```
def add(a, b):

    return a + b
```

Function Without Parameters

```
def show_message():

    print("Welcome to Python!")

show_message()
```

LIST IN PYTHON

Lists are ordered collection of data items.
They store multiple items in a single variable.
List items are separated by commas and enclosed within square brackets [].
Lists are changeable meaning we can alter them after creation.

Example 1:

```
lst1 = [1,2,2,3,5,4,6]

lst2 = ["Red", "Green", "Blue"]

print(lst1)

print(lst2)
```

Output:

```
[1, 2, 2, 3, 5, 4, 6]
['Red', 'Green', 'Blue']
```

Example 2:

```
details = ["Abhijeet", 18, "FYBScIT", 9.8]

print(details)
```

Output:

```
['Abhijeet', 18, 'FYBScIT', 9.8]
```

BASICS OF LIST INDEXING

Indexes start from 0 (not 1).
You use square brackets [] to access elements.

```
my_list = ['apple', 'banana', 'cherry']

print(my_list[0])  # Output: 'apple'

print(my_list[1])  # Output: 'banana'

print(my_list[2])  # Output: 'cherry'
```

Negative Indexing

Negative numbers count from the end of the list.

```
print(my_list[-1])  # 'cherry'

print(my_list[-2])  # 'banana'

print(my_list[-3])  # 'apple'
```

IndexError

Trying to access an index that doesn't exist will raise an error:

```
print(my_list[5])  # IndexError: list index out of range
```

Modifying List Elements Using Index

```
my_list[1] = 'mango'

print(my_list)  # ['apple', 'mango', 'cherry']
```

Using Index in Loops

```
for i in range(len(my_list)):

    print(f"Element at index {i} is {my_list[i]}")
```

List Slicing (a related concept)

```
print(my_list[0:2])   # ['apple', 'mango'] — from index 0 to 1

print(my_list[:])     # full list

print(my_list[::2])   # skip every other element
```

LIST COMPREHENSION

List comprehensions are used for creating new lists from other iterables like lists, tuples, dictionaries, sets, and even in arrays and strings.

Syntax:

```
List = [Expression(item) for item in iterable if Condition]
```

Expression: It is the item which is being iterated.

Iterable: It can be list, tuples, dictionaries, sets, and even in arrays and strings.

Condition: Condition checks if the item should be added to the new list or not.

Example 1: Accepts items with the small letter "o" in the new list

```
names = ["Milo", "Sarah", "Bruno", "Anastasia", "Rosa"]

namesWith_O = [item for item in names if "o" in item]

print(namesWith_O)

Output:

['Milo', 'Bruno', 'Rosa']
```

Example 2: Accepts items which have more than 4 letters

```
names = ["Milo", "Sarah", "Bruno", "Anastasia", "Rosa"]

namesWith_O = [item for item in names if (len(item) > 4)]

print(namesWith_O)
```

Output:

```
['Sarah', 'Bruno', 'Anastasia']
```

LIST METHODS

1.sort()

This method sorts the list in ascending order. The original list is updated

Example 1:

```
colors = ["voilet", "indigo", "blue", "green"]
colors.sort()
print(colors)

num = [4,2,5,3,6,1,2,1,2,8,9,7]
num.sort()
print(num)
```

Output:

```
['blue', 'green', 'indigo', 'voilet']
[1, 1, 2, 2, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.What if you want to print the list in descending order?

We must give reverse=True as a parameter in the sort method.

Example:

```
colors = ["voilet", "indigo", "blue", "green"]
colors.sort(reverse=True)
print(colors)

num = [4,2,5,3,6,1,2,1,2,8,9,7]
num.sort(reverse=True)
print(num)
```

Output:

```
['voilet', 'indigo', 'green', 'blue']
[9, 8, 7, 6, 5, 4, 3, 2, 2, 2, 1, 1]
```

3.The reverse parameter is set to False by default.

reverse()

Example:
```
colors = ["voilet", "indigo", "blue", "green"]
colors.reverse()
print(colors)

num = [4,2,5,3,6,1,2,1,2,8,9,7]
num.reverse()
print(num)
```
Output:
```
['green', 'blue', 'indigo', 'voilet']
[7, 9, 8, 2, 1, 2, 1, 6, 3, 5, 2, 4]
```

4.index()
This method returns the index of the first occurrence of the list item.

Example:
```
colors = ["voilet", "green", "indigo", "blue", "green"]
print(colors.index("green"))

num = [4,2,5,3,6,1,2,1,3,2,8,9,7]
print(num.index(3))
```
Output:
```
1
3
```

5.count()
Returns the count of the number of items with the given value.

Example:
```
colors = ["voilet", "green", "indigo", "blue", "green"]
print(colors.count("green"))

num = [4,2,5,3,6,1,2,1,3,2,8,9,7]
```
Output:
```
2
3
```

6.copy()
Returns copy of the list. This can be done to perform operations on the list without modifying the original list.

Example:
```
colors = ["voilet", "green", "indigo", "blue"]
newlist = colors.copy()
print(colors)
print(newlist)
```
Output:
```
['voilet', 'green', 'indigo', 'blue']
['voilet', 'green', 'indigo', 'blue']
```

7.append():
This method appends items to the end of the existing list.

Example:
```
colors = ["voilet", "indigo", "blue"]
colors.append("green")
print(colors)
```
Output:
```
['voilet', 'indigo', 'blue', 'green']
```

8.insert():
This method inserts an item at the given index. User has to specify index and the item to be inserted within the insert() method.

Example:
```
colors = ["voilet", "indigo", "blue"]
#           [0]         [1]       [2]

colors.insert(1, "green")    #inserts item at index 1
# updated list: colors = ["voilet", "green", "indigo", "blue"]
#       indexs                [0]       [1]      [2]       [3]

print(colors)
```
Output:
```
['voilet', 'green', 'indigo', 'blue']
```

9.extend():
This method adds an entire list or any other collection datatype (set, tuple, dictionary) to the existing list.

Example 1:
```
#add a list to a list
colors = ["voilet", "indigo", "blue"]
rainbow = ["green", "yellow", "orange", "red"]
colors.extend(rainbow)
print(colors)
```
Output:
```
['voilet', 'indigo', 'blue', 'green', 'yellow', 'orange', 'red']
```

10.Concatenating two lists:
You can simply concatenate two lists to join two lists.

Example:
```
colors = ["voilet", "indigo", "blue", "green"]
colors2 = ["yellow", "orange", "red"]
print(colors + colors2)
```
Output:
```
['voilet', 'indigo', 'blue', 'green', 'yellow', 'orange', 'red']
```

TUPLES IN PYTHON

Tuples are ordered collection of data items. They store multiple items in a single variable. Tuple items are separated by commas and enclosed within round brackets (). Tuples are unchangeable meaning we can not alter them after creation.

Example:
```
tuple1 = (1,2,2,3,5,4,6)
tuple2 = ("Red", "Green", "Blue")
print(tuple1)
print(tuple2)
```
Output:
```
(1, 2, 2, 3, 5, 4, 6)
('Red', 'Green', 'Blue')\
```

Tuple Indexes

Each item/element in a tuple has its own unique index. This index can be used to access any particular item from the tuple. The first item has index [0], second item has index [1], third item has index [2] and so on.
Example:
```
country = ("Spain", "Italy", "India",)
#             [0]       [1]       [2]
```

Accessing tuple items:

I. Positive Indexing:

As we have seen that tuple items have index, as such we can access items using these indexes.

Example:

```
country = ("Spain", "Italy", "India",)
#             [0]      [1]      [2]
print(country[0])
print(country[1])
```

Output:

```
Spain
Italy
```

II. Negative Indexing:

Example:

```
country = ("Spain", "Italy", "India", "England", "Germany")
#             [0]      [1]      [2]       [3]        [4]
print(country[-1]) # Similar to print(country[len(country) - 1])
print(country[-3])
```

Output:

```
Germany
India
```

III. Check for item:

We can check if a given item is present in the tuple. This is done using the in keyword.

Example 1:

```
country = ("Spain", "Italy", "India", "England", "Germany")
if "Germany" in country:
    print("Germany is present.")
else:
    print("Germany is absent.")
```

Output:

```
Germany is present.
```

IV. Range of Index:

You can print a range of tuple items by specifying where do you want to start, where do you want to end and if you want to skip elements in between the range.

Syntax:

```
Tuple[start : end : jumpIndex]
```

Note: jump Index is optional. We will see this in given examples.

Example 1: Printing elements within a particular range:

```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey",
"goat", "cow")
print(animals[3:7])     #using positive indexes
print(animals[-7:-2])   #using negative indexes
```

Output:

```
('mouse', 'pig', 'horse', 'donkey')
('bat', 'mouse', 'pig', 'horse', 'donkey')
```

Example 2: Printing all element from a given index till the end

```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey",
"goat", "cow")
print(animals[4:])      #using positive indexes
print(animals[-4:])     #using negative indexes
```

Output:

```
('pig', 'horse', 'donkey', 'goat', 'cow')
('horse', 'donkey', 'goat', 'cow')
```

When no end index is provided, the interpreter prints all the values till the end.

Example 3: printing all elements from start to a given index

```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey",
"goat", "cow")
print(animals[:6])      #using positive indexes
print(animals[:-3])     #using negative indexes
```
Output:
('cat', 'dog', 'bat', 'mouse', 'pig', 'horse')
('cat', 'dog', 'bat', 'mouse', 'pig', 'horse')
When no start index is provided, the interpreter prints all the values from start up to the end index provided.

Example 4: Print alternate values
```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey",
"goat", "cow")
print(animals[::2])     #using positive indexes
print(animals[-8:-1:2]) #using negative indexes
```
Output:
('cat', 'bat', 'pig', 'donkey', 'cow')
('dog', 'mouse', 'horse', 'goat')

Manipulating Tuples

Tuples are immutable, hence if you want to add, remove or change tuple items, then first you must convert the tuple to a list. Then perform operation on that list and convert it back to tuple.
Example:
```
countries = ("Spain", "Italy", "India", "England", "Germany")
temp = list(countries)
temp.append("Russia")       #add item
temp.pop(3)                 #remove item
temp[2] = "Finland"         #change item
countries = tuple(temp)
print(countries)
```
Output:
('Spain', 'Italy', 'Finland', 'Germany', 'Russia')

Tuple methods

As tuple is immutable type of collection of elements it have limited built in methods.They are explained below

count() Method

The count() method of Tuple returns the number of times the given element appears in the tuple.
Syntax:
```
tuple.count(element)
```
Example
```
Tuple1 = (0, 1, 2, 3, 2, 3, 1, 3, 2)
res = Tuple1.count(3)
print('Count of 3 in Tuple1 is:', res)
```
Output
3

index() method

The Index() method returns the first occurrence of the given element from the tuple.
Syntax:
```
tuple.index(element, start, end)
```
Note: This method raises a ValueError if the element is not found in the tuple.
Example
```
Tuple = (0, 1, 2, 3, 2, 3, 1, 3, 2)
res = Tuple.index(3)
print('First occurrence of 3 is', res)
```
Output: 3