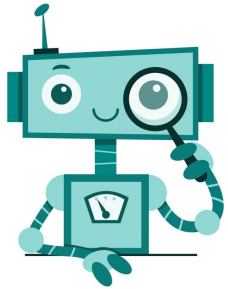
Artificial Intelligence (AI)



Week-4

Agenda

- Tuple
- Dictionary
- Sets
- Conditional Statements

Tuples

- A tuple is a **collection** of objects separated by commas.
- It is similar to a list in terms of **indexing**, **nested objects** and **repetition** but unlike lists a tuple is **immutable.**

```
# An empty tuple
empty_tuple = ()
print (empty_tuple)
```

Output: ()

- make sure to add a **comma** after the element, In case your generating a tuple with a single element.

Access Tuple Items:

Access Tuple Items

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

Negative Indexing

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
cherry
```

Range of Indexes

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

```
('cherry', 'orange', 'kiwi')
```

Access Tuple Items:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

```
('cherry', 'orange', 'kiwi', 'melon', 'mango')
```

Range of Negative Indexes

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

```
('orange', 'kiwi', 'melon')
```

Update Tuples

Add Items

1. **Convert into a list**

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

2. **Add tuple to a tuple**

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

```
('apple', 'banana', 'cherry', 'orange')
```

Change Tuple Values

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

```
('apple', 'kiwi', 'cherry')
```

Update Tuples

Remove Items

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

You can delete the tuple completely

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

Join Tuples

Join Two Tuples

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

Multiply Tuples

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

Basic Tuples Operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in(1,2,3): **print(x)**	1 2 3	Iteration

Built-in Tuple Functions

Sr.No.	Function with Description
1	len(tuple) Gives the total length of the tuple.
2	max(tuple) Returns item from the tuple with max value.
3	min(tuple) Returns item from the tuple with min value.
4	tuple(seq) Converts a list into tuple.

Sequence Types: Range()

range() was a **built-in function** that returned a list in **Python 2** but in python 3 it's a datatype.

```
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 12:01:12) [GCC 4.2.1
(Apple Inc. build 5666) (dot 3)] on darwin Type "help", "copyright",
"credits" or "license" for more information.
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> type(range(10))
<type 'list'>
```

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13) [GCC
4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin Type "help",
"copyright", "credits" or "license" for more information.
>>> range(10)
range(0, 10)
>>> type(range(10))
<class 'range'>
```

Range returns a sequence of numbers starting from zero and increment by 1 by default and stops before the given number.

Syntax:

range (n)
Gives values to 0 to n-1
range(start : stop)
Gives values from the beginning to the n-1
Range (start, stop, step)

```
R = range(10)
R= range(1, 10)
R = range(1, 21,2 )
R= range (20, 1, -2)
```

Mapping Type: Dictionary

- **Dictionary** is a collection of data values.
- It is used to **store data** unlike other Data Types that hold only a single value as an element.
- Dictionary holds **key:value** pair.
- Dictionary can be created by placing a sequence of elements within **curly {} braces**, separated by ‘comma’ and values can be assigned and accessed using **square braces []**
- Dictionary Values are of any **data type** and can be **duplicated,**whereas **keys** are case sensitive, can’t be **repeated** and must be **immutable.**

Syntax:

```
variable = { key1: value1, key2 : Value2}
```

Dictionary can also be created by the built-in function **dict()**. An **empty** dictionary can be created by just placing curly braces**{}**.

```
variable = { }
#we can add values to empty dictionary like:
variable[key1] = value1
variable[key2] = value2
```

```
# Creating an empty Dictionary
variable = {}
print(variable)

# Creating a Dictionary with dict() method
variable = dict({1: 'AI', 2: 'with', 3:'Python'})
print(variable)

# Creating a Dictionary with each item as a Pair
variable = dict([(1, 'AI'), (2, 'with')])
print(variable)
```

Output:

```
{}
{1: 'AI', 2: 'with', 3: 'Python'}
{1: 'AI', 2: 'with'}
```

Nested Dictionary:

- A Nested dictionary can be created in Python by placing the comma-separated dictionaries enclosed within braces.
- **Slicing** Nested Dictionary is not possible.
- We can shrink or grow nested dictionaries as needed.
- **Adding elements** to a **nested** Dictionary can be done in multiple ways.
- One way is to add values one by one.

  Nesteddict[dict][key] = 'value'

- Another way is to add the **whole** dictionary in one go

  Nesteddict[dict] = { 'key': 'value'}

Example:

```
# Python program to print Empty nested dictionary
dict = { 'dict1': { },
        'dict2': { }, 'dict3': { }}
print("Nested dictionary are as follows -")
print(dict)
```

```
Dict = {'Name': 'python', 1: [11, 12, 13]}
print(Dict)
```

Here, we have used the concept of mixed keys, in which keys are not the same. We will extend it and make a nested dictionary with the same keys but different values.

Example:

here we have used the concept of the same keys, in which keys are the same, but the corresponding data values are different.

```
Dict = { 'Dict1': {'Name': 'amina', 'age': '22'},
       'Dict2': {'Name': 'Asim', 'age': '19'}}
print("\n Nested dictionary 2-")
print(Dict)
```

Output:

Nested dictionary 2-

{'Dict1': {'Name': 'amna', 'age': '22'}, 'Dict2': {'Name': 'Asim', 'age': '19'}}

Creating an empty set

Creating an **empty set** is a bit tricky.Empty curly braces **{}** will make an **empty dictionary** in Python. To make a set without any elements, we use the **set()** function **without** any argument.

```
# creation of empty set and dictionary
# initialize a with {}
a = {}
# check data type of a
print(type(a))
# initialize a with set()
a = set()
# check data type of a
print(type(a))
```

Output: <class 'dict'>
<class 'set'>

Adding and Removing elements

It can have any number of items and they may be of different types (integer, float, tuple, string etc.).

We can add and remove elements form the set by:

add(): Adds a given element to a set

clear(): Removes all elements from the set

discard(): Removes the element from the set

remove(): Removes the element from the set

pop(): Returns and removes a random element from the set

```
# set of integers
my_set = {1, 2, 3}
print(my_set)
# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

Output: {1, 2, 3}
{1.0, (1, 2, 3), 'Hello'}

Add Set Items

Add Items

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

```
{'cherry', 'apple', 'orange', 'banana'}
```

Add Any Iterable

```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)
```

Add Sets

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)
```

```
{'cherry', 'papaya', 'pineapple', 'apple', 'mango', 'banana'}
```

Remove Set Items

Remove Item

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

```
{'cherry', 'apple'}
```

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

```
set()
```

```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

```
{'cherry', 'apple'}
```

```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)
print(thisset)
```

```
cherry
{'apple', 'banana'}
```

Combining Sets

Union of set1 and set2 is a set of all elements from both sets.

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
{1, 'b', 2, 'a', 3, 'c'}
```

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

```
{1, 'b', 2, 'a', 3, 'c'}
```

The intersection and difference

Intersection of A and B is a set of elements that are common in both the sets.

```
A = {1, 2, 3, 4, 5}
B = {3, 4, 5, 6, 7, 8}
A.intersection(B)
```

Output: {3, 4, 5}

```
A = {1, 2, 3, 4, 5}
B = {3, 4, 5, 6, 7, 8}
A.difference(B)
B.difference(A)
```

Output: {1, 2}
Output: {6, 7, 8}

Difference of the set B from set A, (A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of elements in B but not in A.

Example:

```
# random dictionary
person = {"name": "Abid", "age": 24,
"Gender": "male"}

fSet = frozenset(person)
print('The frozen set is:', fSet)
```

Output: The frozen set is: frozenset({'name', 'gender', 'age'})

Like normal sets, frozenset can also **perform** different operations like copy, difference, intersection and union.

None Data Type:

None data type is nothing i.e no value is associated with it. To make the value available for the garbage collection.

```
a=None
Print (type(a))
```

Conditional Statement

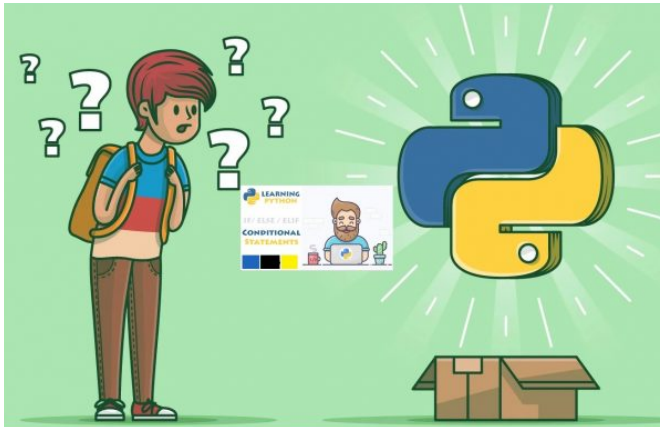
- In Programming Language **Decisions** are taken by using **conditional statements**, in which **Python** evaluates the code to see if it meets the specified conditions.
- **Conditions** are evaluated and processed as True or False.

Conditional Statement:

If the condition is found to be **True**, the program is run as needed and If the condition is found to be **False**, then the statement after the condition is executed.

Conditional Statements:

- if statements
- if-else statements
- elif statements
- Nested if and if-else statements



If Statement:

- **if statement** is one of the most commonly used conditional statements in programming languages.
- It has a code body that only executes if the condition in the if statement is **true**.

Syntax :

```
if expression:
    Statement
```

If the condition is true, the statement will be executed.

(The statement can be a single line or a block of code.)

Example :

```
num = 10
if num > 0:
    print(num, "is a positive number.")
```

OUTPUT: 10 is a positive number.

```
a = 25
b = 100
if b > a:
  print("b is greater than a")
```

OUTPUT: b is greater than a

if-else statement:

This statement deals with both the true and false parts of a given condition. If the condition is true, the statement inside the **if** block is executed and if the condition is false, the statement outside the **if** block is executed.

Syntax :

```
if condition :
    Statement
        #This block execute when the condition is true
else :
    Statement
        #This block execute when the condition is false
```

Example :

```
num = 10
if num >= 1:
    print("Positive number")
else:
     print("Negative number")
```

OUTPUT: Positive number

elif statement:

- **elif Statement** is used to check multiple Conditions, **If** condition is evaluated first
- If it is false, **elif** statement execute and if it is also false, then **else** statement will execute.

Syntax :

```
if condition :
    Statement
elif condition :
    Statement
else:
    Statement
```

elif statements are like **if-else** statement, only difference is **elif statements** evaluate multiple conditions.

Example :

```
num = 5
if num == 0:
    print("Zero")

elif num > 0:
    print("Positive number")

else:
    print("Negative number")
```

OUTPUT: Positive number

Nested if and if-else statements:

Nested if-else statements are statements in which **if** statement or **if-else** statement is present inside another **if** or **if-else** block.

Lab Task -5

Part 1: Tuples

- Create a tuple named **ai_concepts** with elements ‘Machine Learning’, ‘Neural Networks’, ‘Genetic Algorithms’, ‘Natural Language Processing’.
- Find the index of ‘Genetic Algorithms’ in the **ai_concepts** tuple.

Part 2: Dictionaries

- Create a dictionary named **ai_tools** with keys as the AI concepts from **ai_concepts** tuple and values as examples of tools or libraries used in each concept (e.g., ‘Machine Learning’: ‘scikit-learn’).
- Change the tool associated with ‘Neural Networks’ in **ai_tools** to a different one, say ‘TensorFlow’.

Part 3: Sets

- Create a set named **completed_courses** with some AI-related course titles.
- Create another set named **upcoming_courses** with different AI-related course titles.
- Find the courses that are unique to **completed_courses** and not in **upcoming_courses**.

Part 4: Conditional Statements

- Write a series of if statements that classify a student’s grade. If the grade is above 90, print ‘Excellent’; between 80 and 90, print ‘Good’; between 70 and 80, print ‘Average’; below 70, print ‘Needs Improvement’.
- Use a series of if statements to suggest an AI concept to study based on a given interest. For example, if the interest is ‘data’, print ‘Study Machine Learning’.