Data Structures In Python

Topics to cover

Let's Go

- Data Structures
- Lists
- Tuples
- Dictionaries

For the github repositotry visit : Python Repository

Data Structures

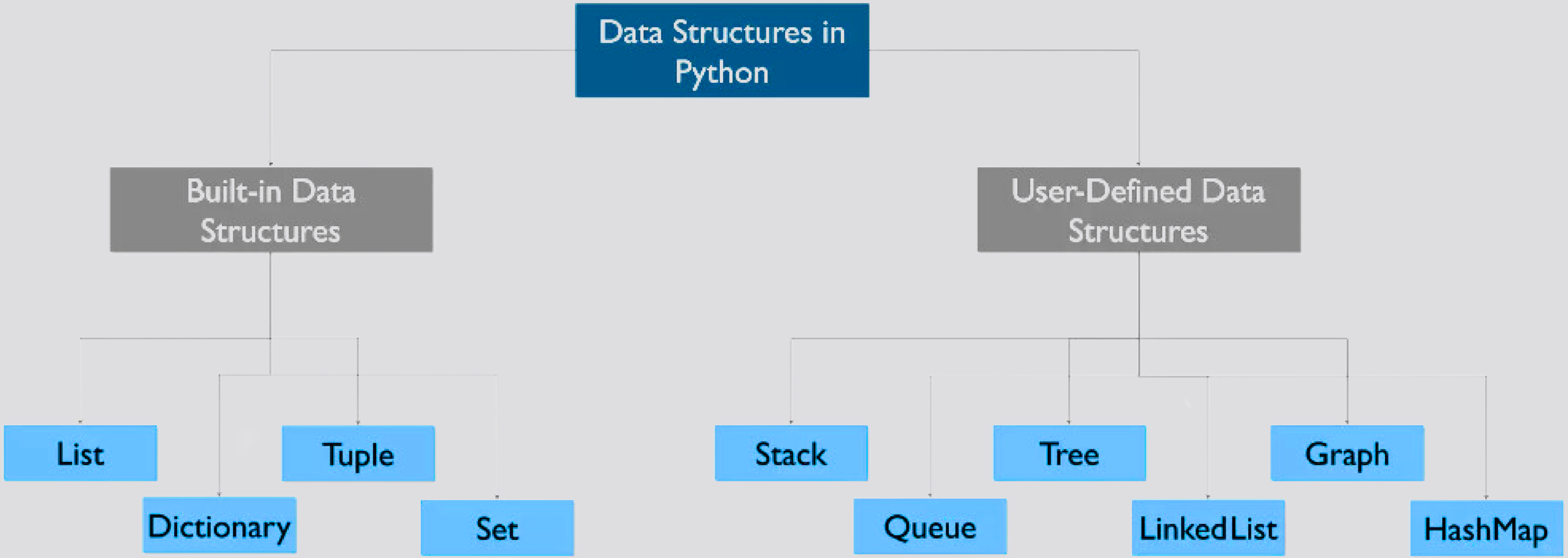
What are Data Structures?

A data structure is a **storage** that is used to **store and organize data**. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.

Why do you need data structures?

A data structure is not only used for organizing the data. It is also used for processing, retrieving, and storing data. There are different basic and advanced types of data structures that are used in almost every program or software system that has been developed. So we must have good knowledge about data structures.

Types of Data Structures in Python



Mutable and Immutable Objects in Python

Immutable Objects : Immutable Objects are of in-built datatypes like int, float, bool, string, Unicode, and tuple. In simple words, an immutable object can’t be changed after it is created.

Mutable Objects : Mutable Objects are of type Python list, Python dictionary, or Python set. Custom classes are generally mutable. mutability refers to the capability of an object to be changed or modified after its creation.

EASY!

LISTS In PYTHON

Lists 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)
```

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

Example 2 :

```
details = ["Abhijeet", 18, "FYBScIT", 9.8]
print(details)
```

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

Note : You can store objects of multiple **datatypes** in a list

LIST METHODS

Python List Methods has multiple methods to work with Python lists, Below we've explained all the methods you can use with Python lists, for example, append(), copy(), insert(), and more.

- **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']
```

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

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

```
['voilet', 'green', 'indigo', 'blue']
['voilet', 'green', 'indigo', 'blue']
```

- **index()** : This method returns the index of the first occurrence of the list item.
- **count()** : Returns the count of the number of items with the given value.
- **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.

It’s solving time

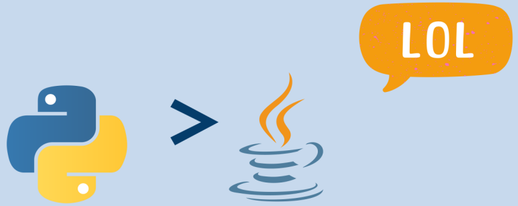
1. Write a Python program to sum all the items in a list.

2. Write a Python program to get the largest number from a list.

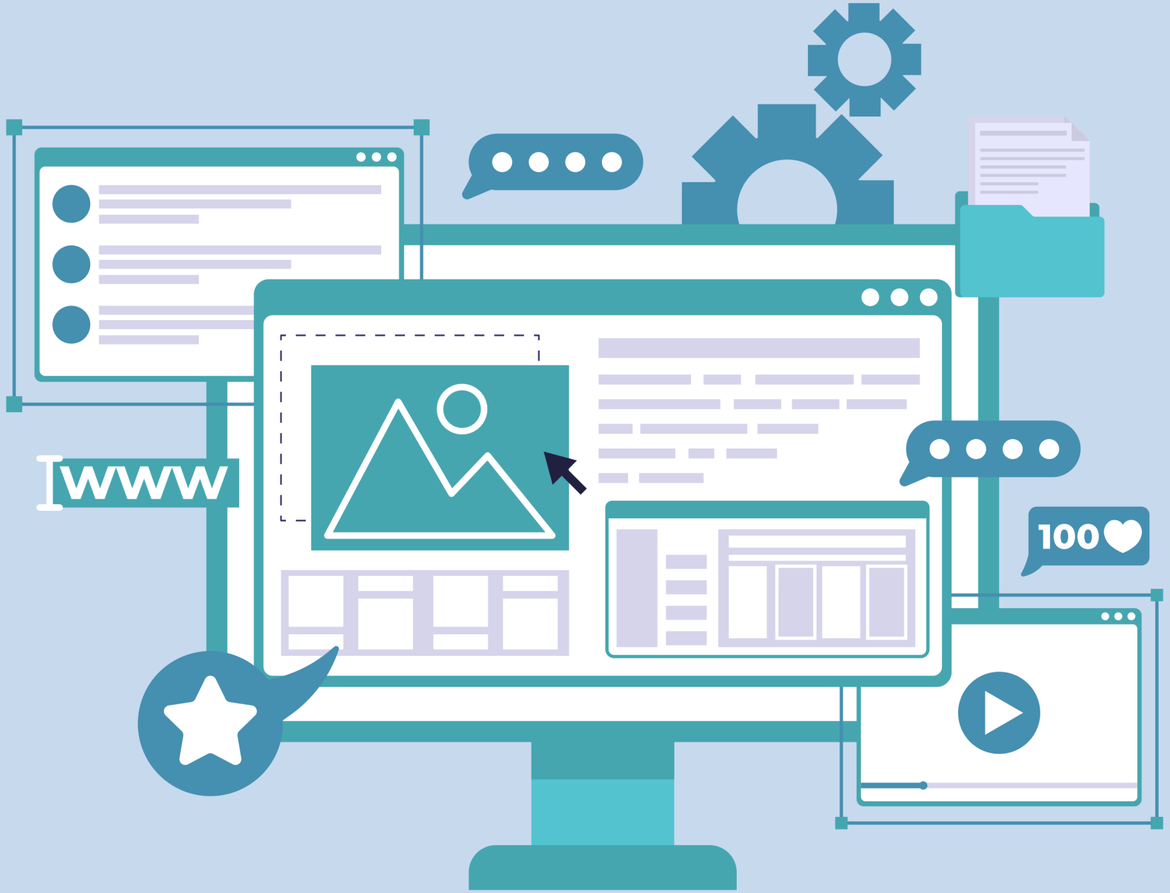
3. Write a Python program to count the number of strings from a given list of strings. The string length is 2 or more and the first and last characters are the same.

4.Write a Python program to remove duplicates from a list.

5. Write a Python program to print the numbers of a specified list after removing even numbers from it.



TUPLES In PYTHON



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 1 :

```
tuple1 = (1,2,2,3,5,4,6)
tuple2 = ("Red", "Green", "Blue")
print(tuple1)
print(tuple2)
```

```
(1, 2, 2, 3, 5, 4, 6)
('Red', 'Green', 'Blue')
```

Example 2:

```python
details = ("Abhijeet", 18, "FYBScIT", 9.8)
print(details)
```

Output:

```
('Abhijeet', 18, 'FYBScIT', 9.8)
```

Float Value

String value

Note : You can store objects of multiple **datatypes** in a list

LOL

- **Are Tuples better than Lists?**
- **Why do we even need Tuples when we have Lists?**

- **sort()** : This method sorts the list in ascending order. The original list is updated.

```
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)
```

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

- **reverse()** : This method reverses the order of the list.

```
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)
```

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

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]
```

- **Positive Indexing :** As we have seen that tuple items have index, as such we can access items using these indexes.

- **Negative Indexing :** Similar to positive indexing, negative indexing is also used to access items, but from the end of the tuple. The last item has index [-1], second last item has index [-2], third last item has index [-3] and so on.

A few examples on how positive and negative indexes work

Example 1: Positive Indexing

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

```
Spain
Italy
India
```

Example 2: Negative Indexing

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

```
Germany
India
Italy
```

Range of index in a Tuple (Tuple Slicing)

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]

Example: 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
```

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

Example: 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
```

```
('cat', 'dog', 'bat', 'mouse', 'pig', 'horse')
('cat', 'dog', 'bat', 'mouse', 'pig', 'horse')
```

Example: 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
```

```
('cat', 'bat', 'pig', 'donkey', 'cow')
('dog', 'mouse', 'horse', 'goat')
```

Operations on 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)
```

```
('Spain', 'Italy', 'Finland', 'Germany', 'Russia')
```

Here we are converting the tuple countries to a list and performing the operations and converting it back to tuples

However, we can directly concatenate two tuples without converting them to list. Because we are not exactly changing the values in a tuple but creating a new tuple.

Example :

```
countries = ("Pakistan", "Afghanistan", "Bangladesh", "ShriLanka")
countries2 = ("Vietnam", "India", "China")
southEastAsia = countries + countries2
print(southEastAsia)
```

Output:

```
('Pakistan', 'Afghanistan', 'Bangladesh', 'ShriLanka', 'Vietnam', 'India', 'China')
```

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)
  ```

- **index() Method:** The Index() method returns the first occurrence of the given element from the tuple.

  Syntax:
  ```
  tuple.index(element, start, end)
  ```

It’s solving time

1 .Write a Python program to convert a tuple to a string.

2. Write a Python program to find repeated items in a tuple.

3. Write a Python program to count the elements in a list until an element is a tuple.

4. Write a Python program to convert a given tuple of positive integers into an integer.

Dictionaries In Python

Python Dictionaries

Dictionaries are ordered collection of data items. They store multiple items in a single variable. Dictionary items are key-value pairs that are separated by commas and enclosed within curly brackets {}.

Example :

```
info = {'name':'Karan', 'age':19, 'eligible':True}
print(info)
```

Output:

```
{'name': 'Karan', 'age': 19, 'eligible': True}
```

Accessing Dictionary items:

I. Accessing single values: Values in a dictionary can be accessed using keys. We can access dictionary values by mentioning keys either in square brackets or by using get method.

Example:

```
info = {'name':'Karan', 'age':19, 'eligible':True}
print(info['name'])
print(info.get('eligible'))
```

```
Karan
True
```

II. Accessing multiple values: We can print all the values in the dictionary using values() method.

Example:

```
info = {'name':'Karan', 'age':19, 'eligible':True}
print(info.values())
```

III. Accessing keys: We can print all the keys in the dictionary using keys() method.

Example:

```
info = {'name':'Karan', 'age':19, 'eligible':True}
print(info.keys())
```

```
dict_keys(['name', 'age', 'eligible'])
```

IV. Accessing key-value pairs: We can print all the key-value pairs in the dictionary using items() method.

Example:

```
info = {'name':'Karan', 'age':19, 'eligible':True}
print(info.items())
```

```
dict_items([('name', 'Karan'), ('age', 19), ('eligible', True)])
```

Dictionary Methods

- **clear() :** The clear() method removes all the elements from a dictionary.
- **copy() :** The copy() method returns a copy of the specified dictionary.
- **pop() :** The pop() method removes the specified item from the dictionary. You need to specify an existing key to use pop()
- **popitem() :** The popitem() method removes the item that was last inserted into the dictionary. **In versions before 3.7, the popitem() method removes a random item**.
- **update() :** The update() method inserts the specified items to the dictionary. The specified items can be a dictionary, or an iterable object with key value pairs.

- **del :** The Python del keyword is used to delete objects.

Syntax :

```
del ep1
print(ep1)
```

This deletes the whole dictionary, and if you print it, throws an error which has to be handled.

```
del ep1[144]
print(ep1)
```

This deletes the particular key-value pair and not the whole dictionary.

- **get() :** The get() method returns the value of the item with the specified key.

```
car = { "brand": "Ford", "model": "Mustang", "year": 1964}

x = car.get("model")
print(x)
```

Output :

```
Mustang
```

It’s solving time

1.Write a Python script to concatenate the following dictionaries to create a new one.

2. Write a Python script to check whether a given key already exists in a dictionary.

3. Write a Python program to sum all the items in a dictionary.

4. Write a Python program to map two lists into a dictionary.

5. Write a Python program to print all distinct values in a dictionary.

Looping Techniques

- Lists

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

```
apple
banana
cherry
```

for loops

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

```
apple
banana
cherry
```

while loops

- Dictionaries

```
car = { "brand": "Ford", "model": "Mustang", "year": 1964}
for x in car:
  print(x)
```

```
brand
model
year
```

Example:

```
for x in car:
  print(car[x])
```

Output:

```
Ford
Mustang
1964
```

Example:

```
for x in car.values():
  print(x)
```

Output:

```
Ford
Mustang
1964
```

Example:

```
for x, y in car.items():
  print(x, y)
```

Output:

```
brand Ford
model Mustang
year 1964
```

IMPORTANT

Functions In Python

Python Functions

A function is a block of code that performs a specific task whenever it is called. In bigger programs, where we have large amounts of code, it is advisable to create or use existing functions that make the program flow organized and neat.

There are two types of functions:

- **Built-in functions:** These functions are defined and pre-coded in python. Some examples of built-in functions are as follows:
  min(), max(), len(), sum(), type(), range(), dict(), list(), tuple(), set(), print(), etc.

- **User-defined functions:** We can create functions to perform specific tasks as per our needs. Such functions are called user-defined functions.

Syntax:

```
def function_name(parameters):
  pass
  # Code and Statements
```

- Create a function using the def keyword, followed by a function name, followed by a paranthesis (()) and a colon(:).
- Any parameters and arguments should be placed within the parentheses.
- Rules to naming function are similar to that of naming variables.
- Any statements and other code within the function should be indented.

Calling a function:

We call a function by giving the function name, followed by parameters (if any) in the parenthesis.

```
def name(fname, lname):
    print("Hello,", fname, lname)

name("Sam", "Wilson")
```

```
Hello, Sam Wilson
```

```
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

Function Arguments

Information can be passed into functions as arguments.
Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Parameter

Arguments

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

There are four types of arguments that we can provide in a function:

1. **Default arguments:** We can provide a default value while creating a function. This way the function assumes a default value even if a value is not provided in the function call for that argument.

```
def name(fname, mname = "Jhon", lname = "Whatson"):
    print("Hello,", fname, mname, lname)

name("Amy")
```

```
Hello, Amy Jhon Whatson
```

2. Keyword arguments: We can provide arguments with key = value, this way the interpreter recognizes the arguments by the parameter name. Hence, the the order in which the arguments are passed does not matter.

```
def name(fname, mname, lname):
    print("Hello,", fname, mname, lname)

name(mname = "Peter", lname = "Wesker", fname = "Jade")
```

```
Hello, Jade Peter Wesker
```

3. Required arguments: In case we don't pass the arguments with a key = value syntax, then it is necessary to pass the arguments in the correct positional order and the number of arguments passed should match with actual function definition.

Example 1: When number of arguments passed does not match to the actual function definition.

```
def name(fname, mname, lname):
    print("Hello,", fname, mname, lname)

name("Peter", "Quill")
```

```
name("Peter", "Quill")\
TypeError: name() missing 1 required positional argument: 'lname'
```

Example 2: When number of arguments passed matches to the actual function definition.

```
def name(fname, mname, lname):
    print("Hello,", fname, mname, lname)

name("Peter", "Ego", "Quill")
```

```
Hello, Peter Ego Quill
```

4. Variable-length arguments: Sometimes we may need to pass more arguments than those defined in the actual function. This can be done using variable-length arguments.

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
This way the function will receive a **tuple of arguments**, and can access the items accordingly:

Example1:

```
def name(*name):
    print("Hello,", name[0], name[1], name[2])

name("James", "Buchanan", "Barnes")
```

```
Hello, James Buchanan Barnes
```

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.
This way the function will receive a **dictionary of arguments**, and can access the items accordingly:

Example 2:

```
def name(**name):
    print("Hello,", name["fname"], name["mname"], name["lname"])

name(mname = "Buchanan", lname = "Barnes", fname = "James")
```

Return Statements

To let a function **return a value**, use the return statement

Example:

```
def func(a):
    return 2*a     #returning twice the value of the passed argument

returnedVal = func(4)  #for integer value
print(returnedVal)

returnedVal = func(5.982975)  #for float value
print(returnedVal)

returnedVal = func("Aman")  #for string value
print(returnedVal)
```

Output :

```
8
11.96595
AmanAman
```

The pass Statement

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the **pass statement** to avoid getting an error.

Example:

```
def myfunction():
  pass
```

The pass statement tells python to avoid this function or the function definition and move on to the next piece of code and do not throw any error.

Let’s solve a few questions

1. Function to find the max of three numbers.

2. Python program to reverse a string.

3. Function to calculate the factorial of a number (a non-negative integer).

4. Function that accepts a string and counts the number of upper and lower case letters.

5. Python function that takes a number as a parameter and checks whether the number is prime or not.

THANK YOU VERY MUCH!

Slides by Aman Singh