

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON
UNIT 3: DATA STRUCTURES IN PYTHON

Python List

- List in python is implemented to store the sequence of various type of data
- A list can be defined as a collection of values or items of different types.
- The items in the list are separated with the comma (,) and enclosed with the square brackets [].

A list can be defined as follows.

1. L1 = ["MMP", 102, "USA"]
2. L2 = [1, 2, 3, 4, 5, 6]
3. L3 = [1, "GAD"]

Accessing List

The elements of the list can be accessed by using the slice operator [].

The index starts from 0 and goes to length - 1.

The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

Consider the following example.

List = [0, 1, 2, 3, 4, 5]

0	1	2	3	4	5
---	---	---	---	---	---

List[0] = 0	List[0:] = [0,1,2,3,4,5]
List[1] = 1	List[:] = [0,1,2,3,4,5]
List[2] = 2	List[2:4] = [2, 3]
List[3] = 3	List[1:3] = [1, 2]
List[4] = 4	List[:4] = [0, 1, 2, 3]
List[5] = 5	

Updating List values

Lists are the most versatile data structures in python since they are mutable and their values can be updated by using the slice and assignment operator.

List = [1, 2, 3, 4, 5, 6]

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

```
print(List)
```

Output : [1, 2, 3, 4, 5, 6]

```
List[2] = 10;
```

```
print(List)
```

Output: [1, 2, 10, 4, 5, 6]

```
List[1:3] = [89, 78]
```

```
print(List)
```

Output : [1, 89, 78, 4, 5, 6]

Deleting List values

The list elements can also be deleted by using the **del** keyword. Python also provides us the `remove()` method if we do not know which element is to be deleted from the list.

Consider the following example to delete the list elements.

```
List = [0,1,2,3,4]
```

```
print(List)
```

Output:

```
[0, 1, 2, 3, 4]
```

```
del List[0]
```

```
print(List)
```

Output:

```
[1, 2, 3, 4]
```

```
del List[3]
```

```
print(List)
```

Output:

```
[1, 2, 3]
```

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

Python List Built-in functions

Python provides the following built-in functions which can be used with the lists.

SN	Function	Description
1	cmp(list1, list2)	It compares the elements of both the lists.
2	len(list)	It is used to calculate the length of the list.
3	max(list)	It returns the maximum element of the list.
4	min(list)	It returns the minimum element of the list.
5	list(seq)	It converts any sequence to the list.

Python List Operations

The concatenation (+) and repetition (*) operator work in the same way as they were working with the strings.

Consider a List l1 = [1, 2, 3, 4] and l2 = [5, 6, 7, 8]

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	L1*2 = [1, 2, 3, 4, 1, 2, 3, 4]
Concatenation	It concatenates the list mentioned on either side of the operator.	l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]
Membership	It returns true if a particular item exists in a particular list otherwise false.	print(2 in l1) prints True.
Iteration	The for loop is used to iterate over the list elements.	for i in l1:

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

		<pre>print(i)</pre> Output 1 2 3 4
Length	It is used to get the length of the list	<code>len(l1) = 4</code>

Python Tuple

- Python Tuple is used to store the sequence of immutable python objects.
- Tuple is immutable and the value of the items stored in the tuple cannot be changed.
- A tuple can be written as the collection of comma-separated values enclosed with the small brackets.

Where use tuple

Using tuple instead of list is used in the following scenario.

1. Using tuple instead of list gives us a clear idea that tuple data is constant and must not be changed.
2. Tuple can simulate dictionary without keys. Consider the following nested structure which can be used as a dictionary.

```
[(101, "CO", 22), (102, "ME", 28), (103, "AE", 30)]
```

3. Tuple can be used as the key inside dictionary due to its immutable nature.

A tuple can be defined as follows.

```
T1 = (101, "Ayush", 22)
```

```
T2 = ("Apple", "Banana", "Orange")
```

Accessing tuple

The indexing in the tuple starts from 0 and goes to `length(tuple) - 1`.

The items in the tuple can be accessed by using the slice operator. Python also allows us to use the colon operator to access multiple items in the tuple.

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

Tuple = (0, 1, 2, 3, 4, 5)

0	1	2	3	4	5
Tuple[0] = 0	Tuple[0:] = (0, 1, 2, 3, 4, 5)				
Tuple[1] = 1	Tuple[:] = (0, 1, 2, 3, 4, 5)				
Tuple[2] = 2	Tuple[2:4] = (2, 3)				
Tuple[3] = 3	Tuple[1:3] = (1, 2)				
Tuple[4] = 4	Tuple[:4] = (0, 1, 2, 3)				
Tuple[5] = 5					

The tuple items can not be deleted by using the del keyword as tuples are immutable. To delete an entire tuple, we can use the del keyword with the tuple name

Consider the following example.

```
tuple1 = (1, 2, 3, 4, 5, 6)
print(tuple1)
del tuple1
print(tuple1)
```

Output:

```
(1, 2, 3, 4, 5, 6)
Traceback (most recent call last):
  File "tuple.py", line 4, in <module>
    print(tuple1)
NameError: name 'tuple1' is not defined
```

Basic Tuple operations

The operators like concatenation (+), repetition (*), Membership (in) works in the same way as they work with the list. Consider the following table for more detail.

Let's say Tuple t = (1, 2, 3, 4, 5) and Tuple t1 = (6, 7, 8, 9) are declared.

Operator	Description	Example
Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	T1*2 = (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

Concatenation	It concatenates the tuple mentioned on either side of the operator.	T1+T2 = (1, 2, 3, 4, 5, 6, 7, 8, 9)
Membership	It returns true if a particular item exists in the tuple otherwise false.	print (2 in T1) prints True.
Iteration	The for loop is used to iterate over the tuple elements.	for i in T1: print(i) Output 1 2 3 4 5
Length	It is used to get the length of the tuple.	len(T1) = 5

Python Tuple inbuilt functions

SN	Function	Description
1	cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
2	len(tuple)	It calculates the length of the tuple.
3	max(tuple)	It returns the maximum element of the tuple.
4	min(tuple)	It returns the minimum element of the tuple.
5	tuple(seq)	It converts the specified sequence to the tuple.

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

List VS Tuple

SN	List	Tuple
1	The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
2	The List is mutable.	The tuple is immutable.
3	The List has the variable length.	The tuple has the fixed length.
4	The list provides more functionality than tuple.	The tuple provides less functionality than the list.
5	The list is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.

Python Set

Unordered collection of various items enclosed within the curly braces.

The elements of the set can not be duplicate.

The elements of the python set must be immutable.

Creating a set

Example 1: using curly braces

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}  
print(Days)  
print(type(Days))
```

Output:

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

```
{'Friday', 'Tuesday', 'Monday', 'Saturday', 'Thursday', 'Sunday', 'Wednesday'}  
<class 'set'>
```

Example 2: using set() method

```
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])  
print(Days)
```

Output:

```
{'Friday', 'Wednesday', 'Thursday', 'Saturday', 'Monday', 'Tuesday', 'Sunday'}
```

Accessing set values

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}  
print(Days)  
print("the set elements ... ")  
for i in Days:  
    print(i)
```

Output:

```
{'Friday', 'Tuesday', 'Monday', 'Saturday', 'Thursday', 'Sunday', 'Wednesday'}  
the set elements ...  
Friday  
Tuesday  
Monday  
Saturday  
Thursday  
Sunday  
Wednesday
```

Removing items from the set

Following methods used to remove the items from the set

1. **discard**
2. **remove**
3. **pop**

- **discard() method**

Python provides **discard()** method which can be used to remove the items from the set.

```
Months = set(["January", "February", "March", "April", "May", "June"])  
print("\nRemoving some months from the set...");  
Months.discard("January");
```


CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

```
Months.discard("May");  
print("\nPrinting the modified set...");  
print(Months)
```

output:

```
{'February', 'January', 'March', 'April', 'June', 'May'}  
Removing some months from the set...  
Printing the modified set...  
{'February', 'March', 'April', 'June'}
```

- **remove() method**

Remove "banana" by using the remove() method:

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

```
thisset.remove("banana")
```

```
print(thisset)
```

output:

```
{"apple", "cherry"}
```

- **pop() method**

the **pop()** method to remove an item, but this method will remove the *last* item. Remember that sets are unordered, so you will not know what item that gets removed.

The return value of the **pop()** method is the removed item.

Note: Sets are *unordered*, so when using the **pop()** method, you will not know which item that gets removed.

Example

Remove the last item by using the **pop()** method:

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

```
x = thisset.pop()
```

```
print(x)
```

```
print(thisset)
```

output:

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

delete the set

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

The `del` keyword will delete the set completely:

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

```
del thisset
```

```
print(thisset)
```

output:

```
File "demo_set_del.py", line 5, in <module>
```

```
    print(thisset) #this will raise an error because the set no longer exists
```

```
NameError: name 'thisset' is not defined
```

Difference between `discard()` and `remove()`

If the key to be deleted from the set using `discard()` doesn't exist in the set, the python will not give the error. The program maintains its control flow.

On the other hand, if the item to be deleted from the set using `remove()` doesn't exist in the set, the python will give the error.

Adding items to the set

- `add()` method
- `update()` method.

Python provides the `add()` method which can be used to add some particular item to the set.

```
Months = set(["January", "February", "March", "April", "May", "June"])
```

```
Months.add("July");
```

```
Months.add("August");
```

```
print(Months)
```

output:

```
{'February', 'July', 'May', 'April', 'March', 'August', 'June', 'January'}
```

```
Months = set(["January", "February", "March", "April", "May", "June"])
```

```
Months.update(["July", "August", "September", "October"]);
```

```
print(Months)
```

output:

```
{'January', 'February', 'April', 'August', 'October', 'May', 'June', 'July', 'September', 'March'}
```

Python set operations (union, intersection, difference and symmetric difference)

In Python, below quick operands can be used for different operations.

| for union.

& for intersection.

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

– for difference
^ for symmetric difference

```
A = {0, 2, 4, 6, 8};  
B = {1, 2, 3, 4, 5};
```

```
# union  
print("Union :", A | B)
```

```
# intersection  
print("Intersection :", A & B)
```

```
# difference  
print("Difference :", A - B)
```

```
# symmetric difference  
print("Symmetric difference :", A ^ B)
```

Output:

```
('Union :', set([0, 1, 2, 3, 4, 5, 6, 8]))  
('Intersection :', set([2, 4]))  
('Difference :', set([8, 0, 6]))  
('Symmetric difference :', set([0, 1, 3, 5, 6, 8]))
```

Built-in Functions with Set

Built-in functions like `all()`, `any()`, `enumerate()`, `len()`, `max()`, `min()`, `sorted()`, `sum()` etc. are commonly used with set to perform different tasks.

Function	Description
all()	Return <code>True</code> if all elements of the set are true (or if the set is empty).
any()	Return <code>True</code> if any element of the set is true. If the set is empty, return <code>False</code> .
enumerate()	Return an enumerate object. It contains the index and value of all the items of set as a pair.
len()	Return the length (the number of items) in the set.
max()	Return the largest item in the set.

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

min()	Return the smallest item in the set.
sorted()	Return a new sorted list from elements in the set(does not sort the set itself).
sum()	Retrun the sum of all elements in the set.

Dictionary

Dictionary is used to implement the key-value pair in python.

The keys are the immutable python object, i.e., Numbers, string or tuple.

Creating the dictionary

The dictionary can be created by using multiple key-value pairs enclosed with the small brackets () and separated by the colon (:).

The collections of the key-value pairs are enclosed within the curly braces {}.

The syntax to define the dictionary is given below.

```
Dict = {"Name": "Ayush", "Age": 22}
```

Accessing the dictionary values

```
Employee = {"Name": "John", "Age": 29, "salary": 25000, "Company": "GOOGLE"}
```

```
print(Employee)
```

output:

```
{'Name': 'John', 'Age': 29, 'salary': 25000, 'Company': 'GOOGLE'}
```

The dictionary values can be accessed in the following way:

```
Employee = {"Name": "John", "Age": 29, "salary": 25000, "Company": "GOOGLE"}
```

```
print("printing Employee data .....")
```

```
print("Name :", Employee["Name"])
```

```
print("Age : ", Employee["Age"])
```

```
print("Salary : ", Employee["salary"])
```

```
print("Company : ", Employee["Company"])
```

Updating dictionary values

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

Dictionary is mutable. We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

```
my_dict = {'name': 'MMP', 'age': 26}
```

update value

```
my_dict['age'] = 27
```

```
print(my_dict)
```

Output: {'age': 27, 'name': 'MMP'}

add item

```
my_dict['address'] = 'Downtown'
```

```
print(my_dict)
```

Output: {'address': 'Downtown', 'age': 27, 'name': 'MMP'}

Deleting elements using del keyword

```
Employee = {"Name": "John", "Age": 29, "salary": 25000, "Company": "GOOGLE"}
```

```
del Employee["Name"]
```

```
del Employee["Company"]
```

```
print("printing the modified information ")
```

```
print(Employee)
```

Output:

```
printing the modified information
{'Age': 29, 'salary': 25000}
```

Dictionary Operations

Below is a list of common dictionary operations:

- create an empty dictionary

```
x = {}
```

- create a three items dictionary

```
x = {"one": 1, "two": 2, "three": 3}
```

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

- access an element

```
x['two']
```

- get a list of all the keys

```
x.keys()
```

- get a list of all the values

```
x.values()
```

- add an entry
- `x["four"]=4`

- change an entry

```
x["one"] = "uno"
```

- delete an entry

```
del x["four"]
```

- remove all items

```
x.clear()
```

- number of items

```
z = len(x)
```

CO: PERFORM OPERATIONS ON DATA STRUCTURES IN PYTHON

- looping over keys

for item in x.keys(): print item

- looping over values

for item in x.values(): print item

Built-in Dictionary functions

The built-in python dictionary methods along with the description are given below.

SN	Function	Description
1	cmp(dict1, dict2)	It compares the items of both the dictionary and returns true if the first dictionary values are greater than the second dictionary, otherwise it returns false.
2	len(dict)	It is used to calculate the length of the dictionary.
3	str(dict)	It converts the dictionary into the printable string representation.
4	type(variable)	It is used to print the type of the passed variable.

Use of Python built-in functions(e.g. type/data conversion functions, math functions , etc)

1. **int(a,base)** : This function converts **any data type to integer**. 'Base' specifies the **base in which string is** if data type is string.

2. **float()** : This function is used to convert **any data type to a floating point number**

Python code to demonstrate Type conversion

initializing string

```
s = "10010"
```

printing string converting to int base 2

```
c = int(s,2)
```

```
print ("After converting to integer base 2 : ", end="")
```

```
print (c)
```

printing string converting to float

```
e = float(s)
```

```
print ("After converting to float : ", end="")
```

```
print (e)
```

Output:

```
After converting to integer base 2 : 18
```

```
After converting to float : 10010.0
```

3. **ord()** : This function is used to convert a **character to integer**.

4. **hex()** : This function is to convert **integer to hexadecimal string**.

5. **oct()** : This function is to convert **integer to octal string**.

initializing integer

```
s = '4'
```

printing character converting to integer

```
c = ord(s)
```

```
print ("After converting character to integer : ",end="")
```

```
print (c)
```

printing integer converting to hexadecimal string

```
c = hex(56)
```

```
print ("After converting 56 to hexadecimal string : ",end="")
```

```
print (c)
```

printing integer converting to octal string

```
c = oct(56)
```

```
print ("After converting 56 to octal string : ",end="")
```

```
print (c)
```

Output:

After converting character to integer : 52

After converting 56 to hexadecimal string : 0x38

After converting 56 to octal string : 0o70

6. tuple() : This function is used to **convert to a tuple**.

7. set() : This function returns the **type after converting to set**.

8. list() : This function is used to convert **any data type to a list type**.

Python code to demonstrate Type conversion

using tuple(), set(), list()

initializing string

s = 'geeks'

printing string converting to tuple

c = tuple(s)

print ("After converting string to tuple : ",end="")

print (c)

printing string converting to set

c = set(s)

print ("After converting string to set : ",end="")

print (c)

printing string converting to list

c = list(s)

print ("After converting string to list : ",end="")

print (c)

Output:

After converting string to tuple : ('g', 'e', 'e', 'k', 's')

After converting string to set : {'k', 'e', 's', 'g'}

After converting string to list : ['g', 'e', 'e', 'k', 's']

9. dict() : This function is used to **convert a tuple of order (key,value) into a dictionary**.

10. str() : Used to **convert integer into a string**.

11. complex(real,imag) : : This function **converts real numbers to complex(real,imag) number**.

Python code to demonstrate Type conversion

using dict(), complex(), str()

initializing integers

a = 1

b = 2

initializing tuple

tup = (('a', 1) ,('f', 2), ('g', 3))

printing integer converting to complex number

```
c = complex(1,2)
print ("After converting integer to complex number : ",end="")
print (c)
```

```
# printing integer converting to string
c = str(a)
print ("After converting integer to string : ",end="")
print (c)
```

```
# printing tuple converting to expression dictionary
c = dict(tup)
print ("After converting tuple to dictionary : ",end="")
print (c)
```

Output:

```
After converting integer to complex number : (1+2j)
After converting integer to string : 1
After converting tuple to dictionary : {'a': 1, 'f': 2, 'g': 3}
```

4.2 User defined functions: Function definition, function calling, function arguments and parameter passing, Return statement, Scope of Variables: Global variable and Local variable.

the user can create its functions which can be called user-defined functions.

In python, we can use **def** keyword to define the function. The syntax to define a function in python is given below.

```
def my_function():
    function code
    return <expression>
```

Function calling

In python, a function must be defined before the function calling otherwise the python interpreter gives an error. Once the function is defined, we can call it from another function or the python prompt. To call the function, use the function name followed by the parentheses.

A simple function that prints the message "Hello Word" is given below.

```
def hello_world():
    print("hello world")

hello_world()
```

Output:

hello world

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses.

```
def hi(name):  
    print(name)
```

```
hi("MMP")
```

Output:

MMP

```
def my_function(fname, lname):  
    print(fname + " " + lname)
```

```
my_function("Purva", "Pawar")
```

Output:

Purva Pawar

Arbitrary Arguments, *args

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.

If the number of arguments is unknown, add a `*` before the parameter name:

```
def my_function(*kids):  
    print("The youngest child is " + kids[1])
```

```
my_function("purva", "sandesh", "jiyansh")
```

Output

The youngest child is sandesh

If the number of keyword arguments is unknown, add a double `**` before the parameter name:

```
def my_function(**kid):  
    print("Her last name is " + kid["lname"])
```

```
my_function(fname = "nitu", lname = "mini")
```

Output

Her last name is mini

Default Parameter Value

If we call the function without argument, it uses the default value:

```
def my_function(country = "Norway"):  
    print("I am from " + country)
```

```
my_function("Sweden")
```

```
my_function("India")
```

```
my_function()
```

```
my_function("Brazil")
```

Output

I am from Sweden

I am from India

I am from Norway

I am from Brazil

Passing a List as an Argument

```
def my_function(food):  
    for x in food:  
        print(x)
```

```
fruits = ["apple", "banana", "cherry"]
```

```
my_function(fruits)
```

Output

apple

banana

cherry

Return statement

```
def my_function(x):
```

```
    return 5 * x
```

```
print(my_function(3))
```

```
print(my_function(5))
```

```
print(my_function(9))
```

Output

15

25

45

Scope of Variables: Global variable and Local variable.

Local variable

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

A variable created inside a function is available inside that function:

```
def myfunc():  
    x = 300  
    print(x)
```

myfunc()

Output

300

Global variable

Global variables are available from within any scope, global and local.

A variable created outside of a function is global and can be used by anyone:

```
x = 300
```

```
def myfunc():  
    print(x)
```

myfunc()

```
print(x)
```

Output

300

300

The `global` keyword makes the variable global.

```
def myfunc():
```

```
    global x
```

```
    x = 300
```

```
myfunc()
```

```
print(x)
```

Output

300

Modules: Writing modules

Shown below is a Python script containing the definition of `SayHello()` function. It is saved as **hello.py**.

Example: hello.py

```
def SayHello(name):  
    print("Hello {}! How are you?".format(name))  
    return
```

importing modules

```
>>> import hello
```

```
>>> hello.SayHello("purva")
```

Output

Hello purva! How are you?

importing objects from modules

To import only parts from a module, by using the `from` keyword.

The module named `mymodule` has one function and one dictionary:

```
def greeting(name):  
    print("Hello, " + name)
```

```
person1 = {  
    "name": "John",  
    "age": 36,  
    "country": "Norway"  
}
```

Import only the person1 dictionary from the module:

```
from mymodule import person1
```

```
print (person1["age"])
```

Output:

Python built-in modules(e.g. Numeric and Mathematical module, Functional programming module)

Python - Math Module

```
>>> import math
>>> math.pi
3.141592653589793

>>> math.log(10)
2.302585092994046

>> math.sin(0.5235987755982988)
0.49999999999999994
>>> math.cos(0.5235987755982988)
0.8660254037844387
>>> math.tan(0.5235987755982988)
0.5773502691896257

>>> math.radians(30)
0.5235987755982988
>>> math.degrees(math.pi/6)
29.999999999999996
```

Namespace and Scoping.

- A namespace is a mapping from names to objects.
 - Python implements namespaces in the form of dictionaries.
 - It maintains a name-to-object mapping where names act as keys and the objects as values.
 - Multiple namespaces may have the same name but pointing to a different variable.

 - A scope is a textual region of a Python program where a namespace is directly accessible.
-
- Local scope
 - Non-local scope
 - Global scope
 - Built-ins scope

1. **The local scope.** The local scope is determined by whether you are in a class/function definition or not. Inside a class/function, the local scope refers to the names defined inside them. Outside a class/function, the local scope is the same as the global scope.
2. **The non-local scope.** A non-local scope is midway between the local scope and the global scope, e.g. the non-local scope of a function defined inside another function is the enclosing function itself.
3. **The global scope.** This refers to the scope outside any functions or class definitions. It also known as the module scope.
4. **The built-ins scope.** This scope, as the name suggests, is a scope that is built into Python. While it resides in its own module, any Python program is qualified to call the names defined here without requiring special access.

var1 is in the global namespace

var1 = 5

def some_func():

var2 is in the local namespace

var2 = 6

def some_inner_func():

var3 is in the nested local

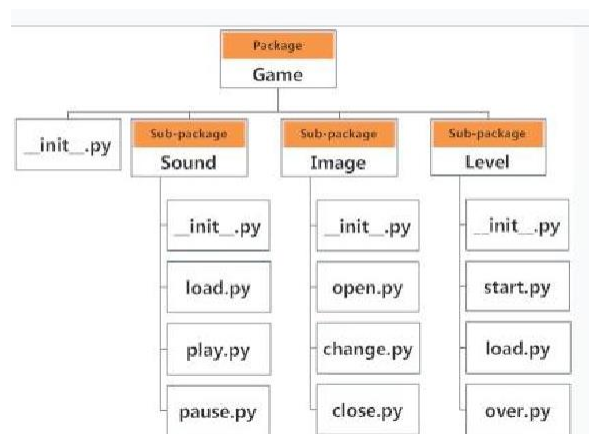
namespace

var3 = 7

Python Packages : Introduction

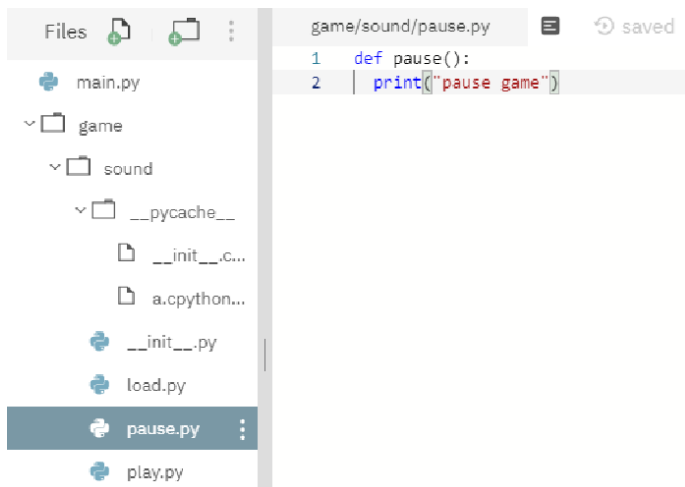
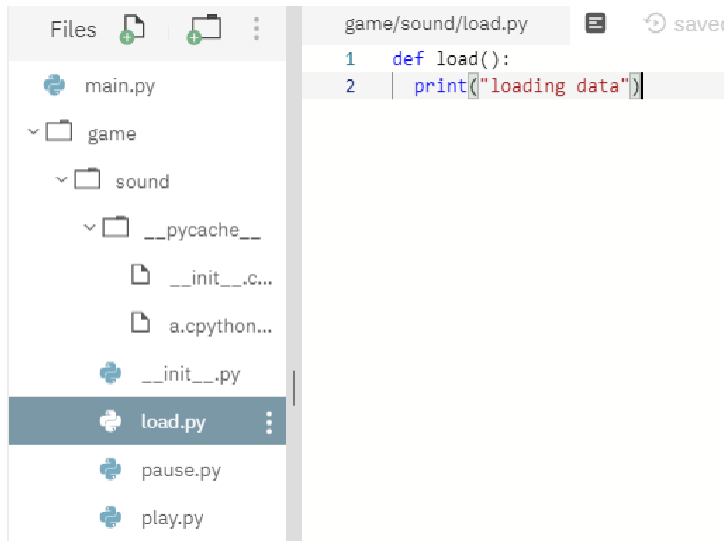
Python has packages for directories and [modules](#) for files. As a directory can contain sub-directories and files, a Python package can have sub-packages and modules.

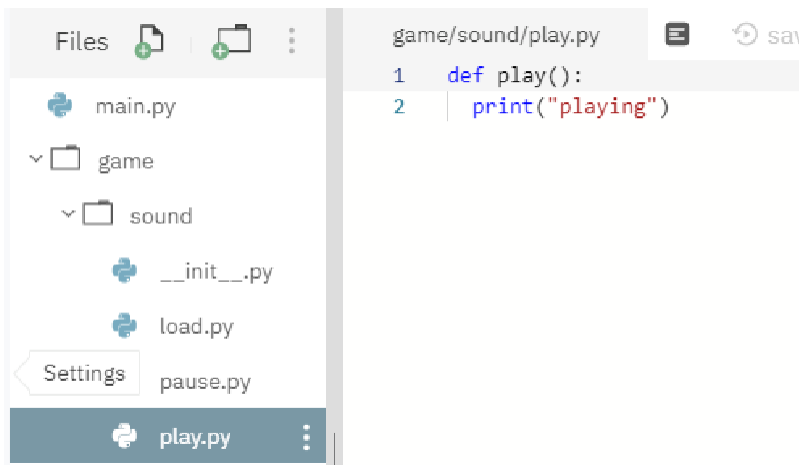
A directory must contain a file named `__init__.py` in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.



Steps:

- First create folder game.
- Inside it again create folder sound.
- Inside sound folder create load.py file.
- Inside sound folder create pause.py file.
- Inside sound folder create play.py file.
- Import package game and subpackage sound(files:load,pause,play)





Importing module from a package

`import game.sound.load`

Now if this module contains a [function](#) named `load()`, we must use the full name to reference it.

`game.sound.load.load()`

Math package:

```
>>> import math
```

```
>>> math.pi
```

```
3.141592653589793
```

sin, cos and tan ratios for the angle of 30 degrees (0.5235987755982988 radians):

```
>>>math.sin(0.5235987755982988)
```

```
0.49999999999999994
```

```
>>>math.cos(0.5235987755982988)
```

```
0.8660254037844387
```

```
>>>math.tan(0.5235987755982988)
```

```
0.5773502691896257
```

NumPy is a python library used for working with arrays.

NumPy stands for Numerical Python.

Why Use NumPy ?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

- Install it using this command:

```
C:\Users\Your Name>pip install numpy
```

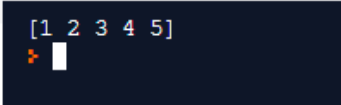
Import NumPy

```
main.py  saved
1  import numpy
2
3  arr = numpy.array([1, 2, 3, 4, 5])
4
5  print(arr)
```

```
[1 2 3 4 5]
>
```

Use a tuple to create a NumPy array:

```
main.py  [icon]  ↻ saving...
1  import numpy as np
2
3  arr = np.array((1, 2, 3, 4, 5))
4
5  print(arr)
6
```



The terminal output shows the array [1 2 3 4 5] on a dark background with a cursor at the end of the line.

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Create a 0-D array with value 42

```
main.py  [icon]  ↻ saving...
1  import numpy as np
2
3  arr = np.array(42)
4
5  print(arr)
```



The terminal output shows the value 42 on a dark background with a cursor at the end of the line.

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

Create a 1-D array containing the values 1,2,3,4,5:

```
main.py  saving...
1  import numpy as np
2
3  arr = np.array([1, 2, 3, 4, 5])
4
5  print(arr)
```

```
[1 2 3 4 5]
```

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
main.py  saving...
1  import numpy as np
2
3  arr = np.array([[1, 2, 3], [4, 5, 6]])
4
5  print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
main.py  saved
1  import numpy as np
2
3  arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
4
5  print(arr)
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

SciPy package

SciPy, pronounced as Sigh Pi, is a scientific python open source, distributed under the BSD licensed library to perform Mathematical, Scientific and Engineering Computations..

The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation.

SciPy Sub-packages

- SciPy consists of all the numerical code.

- SciPy is organized into sub-packages covering different scientific computing domains. These are summarized in the following table –

<u>scipy.cluster</u>	Vector quantization / Kmeans
<u>scipy.constants</u>	Physical and mathematical constants
<u>scipy.fftpack</u>	Fourier transform
<u>scipy.integrate</u>	Integration routines
<u>scipy.interpolate</u>	Interpolation
<u>scipy.io</u>	Data input and output
<u>scipy.linalg</u>	Linear algebra routines
<u>scipy.ndimage</u>	n-dimensional image package
<u>scipy.odr</u>	Orthogonal distance regression
<u>scipy.optimize</u>	Optimization
<u>scipy.signal</u>	Signal processing
<u>scipy.sparse</u>	Sparse matrices
<u>scipy.spatial</u>	Spatial data structures and algorithms
<u>scipy.special</u>	Any special mathematical functions

<u>scipy.stats</u>	Statistics
--------------------	------------

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+ SciPy makes use of Matplotlib.

Pandas is used for data manipulation, analysis and cleaning. Python pandas is well suited for different kinds of data, such as:

- Tabular data with heterogeneously-typed columns
- Ordered and unordered time series data
- Arbitrary matrix data with row & column labels

Any other form of observational