Kmit

TRISHUL

Modules

- **Content**
- Introduction
- Importing module
- Math module
- Random module
- Packages
- Composition

Introduction

- A module is a python object with arbitrarily named attributes that you can bind and reference.
- A module is a file consisting of python code.
- A module can define functions, classes and variables.
- A module can also include runnable code
- Modules are python .py files that consist of python code.
- Any file can be referenced as a module.
- Ex:hello.py has module name of hello that can imported into another python files.
- There are a number of modules that are built into the "Python Standard Library", which contains many modules that provide access to system functionality.
- The "Python Standard Library " is part of every Python installation.

Importing module

- To make use of the functions in a module, we will need to import the module with an "**import**" statement.
- An import statement is made up of the import keyword along with the name of the module.
- In a python file this will be declared at the top of the code.
- Ex:
- supposed to generate random numbers we should import "random" module
- #import module
- import random
- for i in range(10):
- print(random.randint(1,25))

- Ex2:
- we can constant "pi" from math to our program
- import random,math
- for i in range(5):
- print(random.randint(1,25))
- print(math.pi)
- The import statement allows you to import one or more modules into your program.
- "Using from…..import
- To refer to items from a module within your program's namespace, you can use the from..import statement.
- Import statement take in references to everything defined within the module by using an asterik(*)as a wildcard.
- Ex3:
- from random import randint
- for i in range(10):
- print(randint(1,25))

Aliasing Modules

- It is possible to modify the names of modules and their functions within Python by using the keyword "as".
- Syntax:
- Import [module] as [another_name]
- Ex:
- import random as r
- for k in range(5):
- print(r.randint(1,20))
- Making use of modules allows us to make our programs more robust and powerful as we're leveraging existing code.
- We can also create our modules for ourselves and for other programmers to use in future programs.

Random module

- The random module provides access to functions that support many operations.
- Perhaps the most important thing is that it allows you to generate random numbers.
- **When to use it?**
- We want the computer to pick a random number in a given range.
- Pick a random element from a list, pick a random card from a deck, flip a coin etc.
- When making your password database more secure or powering a random page feature of your website.

Random functions

- The Random module contains some very useful functions “randint”.
- If we wanted a random integer, we can use the randint function Randint accepts two parameters: a lowest and a highest number.
- Generate integers between 1,5. The first value should be less than the second.

- Ex:
- import random
- print (random.randint(0, 5))

- **Random**
- If you want a larger number, you can multiply it. For example, a random number between 0 and 100:

- Ex:
- Import random
- print(random.random() * 100)

Choice

- Generate a random value from the sequence.
- Ex: import random
- myList = [2, 109, False, 10, "Lorem", 482, "Ipsum"]
- c1=random.choice(myList)
- Print(c1)
- **Shuffle**
- The shuffle function, shuffles the elements in list in place, so they are in a random order.
- Ex:
- from random import shuffle
- x = [[i] for i in range(10)]
- Shuffle(x)
- print((x))

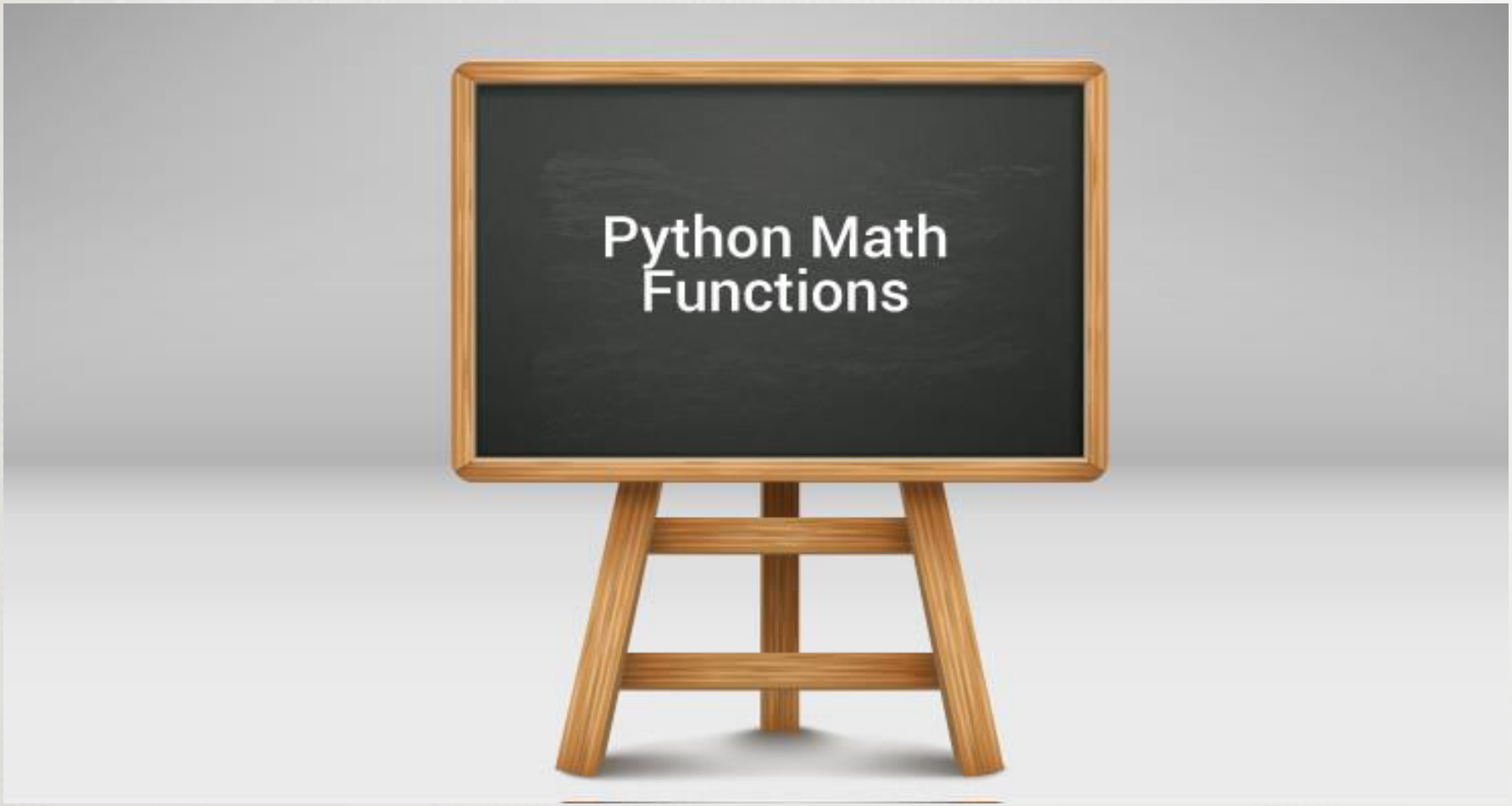
Randrange

- Generate a randomly selected element from range(start, stop, step) random.randrange(start, stop[, step])
- Randrange
- Generate a randomly selected element from range(start, stop, step) random.randrange(start, stop[, step])
- Ex:
- #randomrange
- import random
- for i in range(3):
- print (random.randrange(0, 101))

Kmit

TRISHUL

Math module



- The math module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math.
- Ex:
- #math module
- #Square root calculation
- import math
- print(math.sqrt(4))

- Here is the list of all the functions and attributes defined in math module with a brief explanation of what they do.
- List of Functions in Python Math Module

Function	Description
ceil(x)	Returns the smallest integer greater than or equal to x.
copysign(x, y)	Returns x with the sign of y
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
pow(x, y)	Returns x raised to the power y
sqrt(x)	Returns the square root of x
acos(x)	Returns the arc cosine of x
asin(x)	Returns the arc sine of x


```
# Fractional number using floor and ceil.
n = 100.7
# Absolute value.
print(math.floor(n))
print(math.ceil(n))


# sum of list using sum and fsum
values = [0.9999999, 1, 2, 3]
# Sum values in list.
r = sum(values)
print(r)
# Sum values with fsum.
r = math.fsum(values)
print(r)
```

Ex3:

```
import math


# Use math.pow method.
a = math.pow(2, 3)
# Use operator.
b = 2 ** 3


# Print results.
print(a)
print(b)
```

- Ex4:
- #Python program that uses math.e, pi

- import math

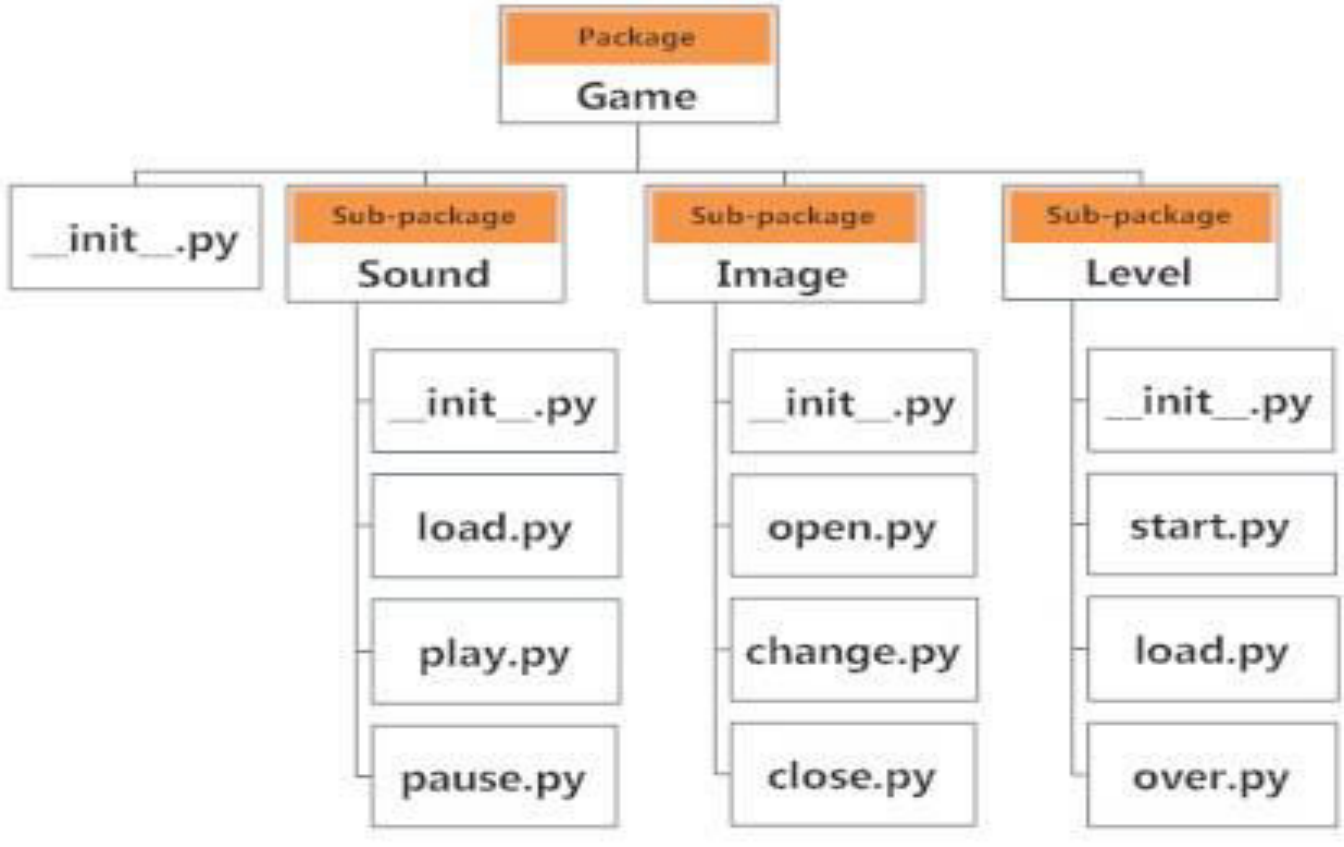
- # This returns the value of e.
- print(math.e)

- # And this is pi.
- print(math.pi)

Packages

Python packages

- We use a well-organized hierarchy of directories for easier access.
- Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory.
- Analogous to this, Python has packages for directories and modules for files.
- As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages.
- This makes a project (program) easy to manage and conceptually clear.


Package
Game
__init__.py
Sub-package
Sound
Sub-package
Image
Sub-package
Level
__init__.py
load.py
play.py
pause.py
__init__.py
open.py
change.py
close.py
__init__.py
start.py
load.py
over.py

- **Importing module from a package**
- We can import modules from packages using the dot (.) operator.
- Ex:
- import Game.Level.start

User Defined Module

- User defined modules and importing into other file
- Module arithematic.py
- #create a module
- def add(x, y):
- """"""""""
- return x + y
- def division(x, y):
- """"""""""
- return x / y
- def multiply(x, y):
- """"""""""
- return x * y
- def subtract(x, y):
- """"""""""

- import arithematic
- print (arithematic.add(5, 8))
- print (arithematic.subtract(10, 5))
- print (arithematic.division(2, 7))
- print (arithematic.multiply(12, 6))

User Defined Module

- Create a with factorial, even, perfect of given number and the functions should return a value to function call

Solution

```
def fact(n):
    f=1
    for i in range(1,n+1):
        f=f*i
    return f
def even(n):
    if(n%2==0):
        return 0
    else:
        return 1
```


```
def perfect(n):

    s=0
    for k in range(1,n):
        if(n%k==0):
            s=s+k
    if(s==n):
        return 1
    else:
        return 0
```


```
#importing user module
import usermodule as f
num=int(input("enter the number:"))
#print(f(num))
print("factorial of given number:")
print(f.fact(num))
res1=f.even(num)
if(res1==0):
    print("Even")
else:
    print("odd")
res=f.perfect(num)
if(res==1):
    print("perefct")
else:
    print("not a perect number")
```

Output:

```
enter the number:5
factorial of given number:
120
odd
not a perect number
```

Composition

- Function composition is a way of combining functions such that the result of each function is passed as the argument of the next function.
- For example, the composition of two functions f and g is denoted f(g(x)).
- x is the argument of g, the result of g is passed as the argument of f and the result of the composition is the result of f.

Ex:

```
#composition
def compose2(f, g):
        return lambda x: f(g(x))
def double(x):
        return x * 2
def inc(x):
        return x + 1
inc_n_d=compose2(double,inc)
print(inc_n_d(10))
```

Composing *n* functions

- Now that we know how to compose two functions, it would be interesting to generalize it to accept *n* functions.
- Since the solution is based on compose2, let's first look at the composition of three functions using compose2.

- Ex:
- #composition
- def compose2(f, g):
- return lambda x: f(g(x))
- def double(x):
- return x * 2
- def inc(x):
- return x + 1
- def dec(x):
- return x - 1
- inc_n_d_dc=compose2(compose2(dec,double),inc)
- print(inc_n_d_dc(10))

- Ex2:
- import functools
- def compose(*functions):
- return functools.reduce(lambda f, g: lambda x: f(g(x)), functions, lambda x: x)
- inc_double_and_dec = compose(dec, double, inc)
- print(inc_double_and_dec(10))