Python Functions

What is a function in Python?

- Is a group of related statements that perform a specific task
- Help break our program into smaller and modular chunks
- As program grows larger and larger, functions make it more organized and manageable
- Furthermore, it avoids repetition and makes code reusable

Syntax of Function

```
def function_name(parameters):
        """docstring"""
        statement(s)
```

Example of a function

```
#Function Definition
def greet(name):
        """This function greets to
        the person passed in as
        parameter"""
        print("Hello, " + name + ". Good morning!")

greet("Ram") #Function Call
```

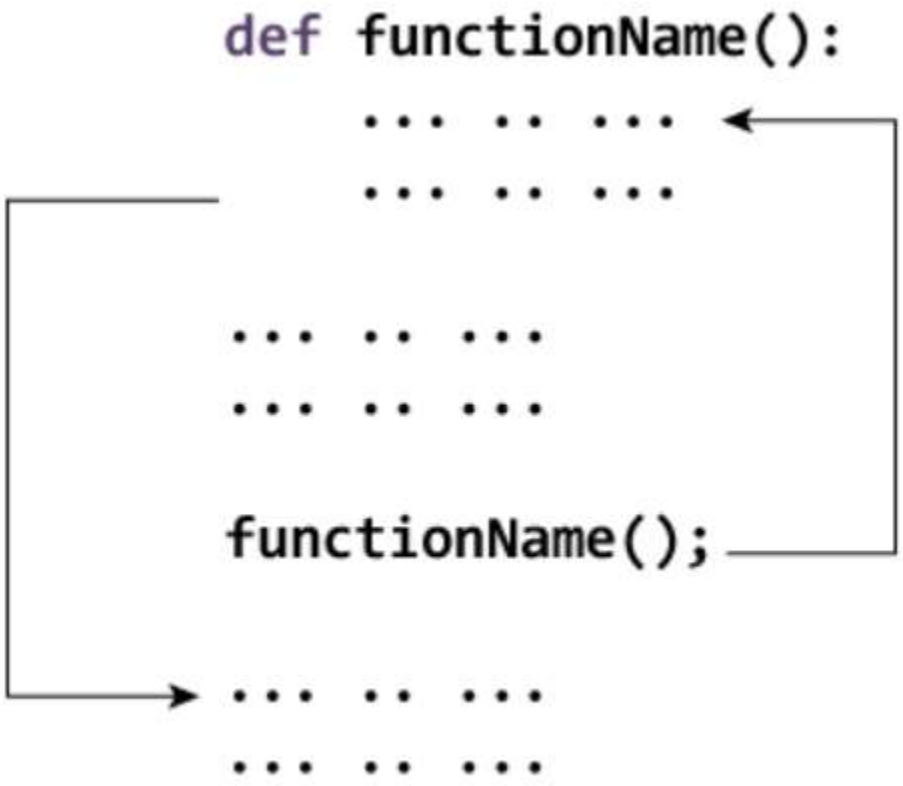
Docstring

```
#Function Definition
def greet(name):
        """This function greets to
        the person passed in as
        parameter"""
        print("Hello, " + name + ". Good morning!")

print(greet.__doc__)
greet("Ram") #Function Call
```

The return statement

```
def absolute_value(num):
        """This function returns the absolute
        value of the entered number"""

        if num >= 0:
                return num
        else:
                return -num

print(absolute_value(2))

print(absolute_value(-4))

print(absolute_value(0))
```

How Function works in Python?



Scope and Lifetime of variables

- Scope of a variable is the portion of a program where the variable is recognized
- Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.
- Lifetime of a variable is the period throughout which the variable exits in the memory.
- The lifetime of variables inside a function is as long as the function executes.
- They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Example

```
def my_func():
        x = 10
        print("Value inside function:",x)

x = 20
my_func()
print("Value outside function:",x)
```

Types of Functions

- Can be divided into the following two types:
  - Built-in functions: Functions that are built into Python
  - User-defined functions: Functions defined by the users themselves

Arguments

```
def greet(name,msg):
    """This function greets to
    the person with the provided message"""
    print("Hello",name + ', ' + msg)

greet("Maan Bahadur","Good morning!")
```

Python Default Arguments

```
def greet(name, msg = "Good morning!"):
    """
    This function greets to
    the person with the
    provided message.

    If message is not provided,
    it defaults to "Good
    morning!"
    """

    print("Hello",name + ', ' + msg)

greet("Ram Bahadur")
greet("Shyam Bahadur","How do you do?")
```

Python Keyword Arguments

```
def greet(name, msg = "Good morning!"):
    """
    This function greets to
    the person with the
    provided message.

    If message is not provided,
    it defaults to "Good
    morning!"
    """

    print("Hello",name + ', ' + msg)

greet(name = "Ram Bahadur",msg = "How do you do?")

greet(msg = "How do you do?",name = "Shyam Bahadur")

greet("Hari Bahadur",msg = "How do you do?")
```

Python Arbitrary Arguments

```
def greet(*names):
    """This function greets all
    the person in the names tuple."""

    # names is a tuple with arguments
    for name in names:
        print("Hello",name)

greet("Gita")
greet("Ram","Shyam","Hari","Sita")
```

Python Recursion

- What is recursion in Python?
- Python Recursive Function
- Advantages of Recursion
- Disadvantages of Recursion

What is recursion in Python?

- Recursion is the process of defining something in terms of itself
- A physical world example would be to place two parallel mirrors facing each other
- Any object in between them would be reflected recursively

Python Recursive Function

```
# An example of a recursive function to
# find the factorial of a number

def calc_factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * calc_factorial(x-1))

num = 4
print("The factorial of", num, "is", calc_factorial(num))
```

```
calc_factorial(4)                    # 1st call with 4
4 * calc_factorial(3)                # 2nd call with 3
4 * 3 * calc_factorial(2)            # 3rd call with 2
4 * 3 * 2 * calc_factorial(1)        # 4th call with 1
4 * 3 * 2 * 1                             # return from 4th call as number=1
4 * 3 * 2                                  # return from 3rd call
4 * 6                                       # return from 2nd call
24                                          # return from 1st call
```

Advantages of Recursion

- Make the code look clean and elegant
- A complex task can be broken down into simpler sub-problems using recursion

Disadvantages of Recursion

- Sometimes the logic behind recursion is hard to follow through
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time
- Recursive functions are hard to debug

Python Anonymous/Lambda Function

- What are lambda functions in Python?
- How to use lambda Functions in Python?
  - Syntax of Lambda Function in python
  - Example of Lambda Function in python
  - Use of Lambda Function in python

What are lambda functions in Python?

- In Python, anonymous function is a function that is defined without a name.
- While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.
- Hence, anonymous functions are also called lambda functions.

How to use lambda Functions in Python?

- Syntax of Lambda Function in python

```
lambda arguments: expression
```

Example of Lambda Function in python

```
# Program to show the use of lambda functions

double = lambda x: x * 2
square= lambda y:y**2



print(double(5))
print(square(4))
```

Use of Lambda Function in python

```
my_list = [1, 5, 4, 6, 8, 11, 3, 12]

odd_list = list(filter(lambda x: (x%2 == 0) , my_list))
even_list= list(filter(lambda x:(x%2!=0),my_list))


print(my_list)
print(odd_list)
print(even_list)
```

```python
# Program to double each item in a list using map()

my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(map(lambda x: x * 2 , my_list))

print(my_list)
print(new_list)
```

Python Global, Local and Nonlocal variables

- Global Variables in Python
- Local Variables in Python
- Global and Local Variables Together
- Nonlocal Variables in Python

Global Variables in Python

```
x = "global"

def foo():
    print("x inside :", x)

foo()
print("x outside:", x)
```

```
x = "global"

def foo():
    x = x * 2
    print(x)
foo()
```

Local Variables in Python

```
def foo():
    y = "local"
    print(y)

foo()
```

Global and Local Variables Together

```
x = "global"

def foo():
    global x
    y = "local"
    x = x * 2
    print(x)
    print(y)

foo()
```

```
x = 5

def foo():
    x = 10
    print("local x:", x)

foo()
print("global x:", x)
```

Nonlocal Variables in Python

```
x = 0
def outer():
    x = 1
    def inner():
        nonlocal x
        x = 2
        print("inner:", x)

    inner()
    print("outer:", x)

outer()
print("global:", x)
```

Python Global Keyword

- Python global Keyword
  - Rules of global Keyword
  - Use of global Keyword (With Example)
- Global Variables Across Python Modules
- Global in Nested Functions in Python

Python global Keyword

- The basic rules for global keyword in Python are:
  - When we create a variable inside a function, it's local by default
  - When we define a variable outside of a function, it's global by default. We don't have to use global keyword.
  - We use global keyword to read and write a global variable inside a function.
  - Use of global keyword outside a function has no effect

```
c = 1 # global variable

def add():
    print(c)

add()
```

```
c = 1 # global variable

def add():
    c = c + 2 # increment c by 2
    print(c)

add()
```

```
c = 0 # global variable

def add():
    global c
    c = c + 2 # increment by 2
    print("Inside add():", c)

add()
print("In main:", c)
```

Global Variables Across Python Modules

```
#value.py
a=0
b="empty"
```

```
#update.py
import value

value.a = 10
value.b = "alphabet"
```

```
import value
import update

print(value.a)
print(value.b)
```

Global in Nested Functions

```
def foo():
    x = 20

    def bar():
        global x
        x = 25

    print("Before calling bar: ", x)
    print("Calling bar now")
    bar()
    print("After calling bar: ", x)

foo()
print("x in main : ", x)
```

Python Modules

- What are modules in Python?
- How to import modules in Python?
  - Python import statement
  - Import with renaming
  - Python from...import statement
  - Import all names
- Python Module Search Path
- Reloading a module
- The dir() built-in function

What are modules in Python?

- File containing Python statements and definitions
  - Example: example.py, is called a module
- Use to break down large programs into small manageable and organized files.
- Provide reusability of code
- Can define our most used functions in a module and import it, instead of copying their definitions into different programs

How to import modules in Python?

- Python import statement
- Import with renaming
- Python from...import statement
- Import all names

Python import statement

```
# import statement example
# to import standard module math

import math
print("The value of pi is", math.pi)
```

Import with renaming

```
# import module by renaming it

import math as m
print("The value of pi is", m.pi)
```

Python from...import statement

```
# import only pi from math module

from math import pi
print("The value of pi is", pi)
```

Import all names

```
# import all names from the standard module math

from math import *
print("The value of pi is", pi)
print("The value of e is", e)
```

Python Module Search Path

- While importing a module, Python looks at several places.
- Interpreter first looks for a built-in module then (if not found) into a list of directories defined in sys.path.
- The search is in this order.
  - The current directory.
  - PYTHONPATH (an environment variable with a list of directory).
  - The installation-dependent default directory.

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import sys
>>> sys.path
['', 'C:\\Users\\Cab\\AppData\\Local\\Programs\\Python\\Python36\\Lib\\idlelib',
 'C:\\Users\\Cab\\AppData\\Local\\Programs\\Python\\Python36\\python36.zip', 'C:
\\Users\\Cab\\AppData\\Local\\Programs\\Python\\Python36\\DLLs', 'C:\\Users\\Cab
\\AppData\\Local\\Programs\\Python\\Python36\\lib', 'C:\\Users\\Cab\\AppData\\Lo
cal\\Programs\\Python\\Python36', 'C:\\Users\\Cab\\AppData\\Local\\Programs\\Pyt
hon\\Python36\\lib\\site-packages']
>>>
```

Reloading a module

```
#function.py
print("Funtion module")
```

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import function
Funtion module
>>> import function
>>> import function
>>> import imp
>>> imp.reload(function)
Funtion module
<module 'function' from 'C:\\Users\\Cab\\AppData\\Local\\Programs\\Python\\Pytho
n36\\function.py'>
>>>
```

The dir() built-in function

- We can use the dir() function to find out names that are defined inside a module.

```python
#function.py
a=40
b="Ram Bahadur"
def test():
    c=65.5
    print(c)
print(a,b)
test()
```

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import function
40 Ram Bahadur
65.5
>>> dir(function)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'a', 'b', 'test']
>>>
```

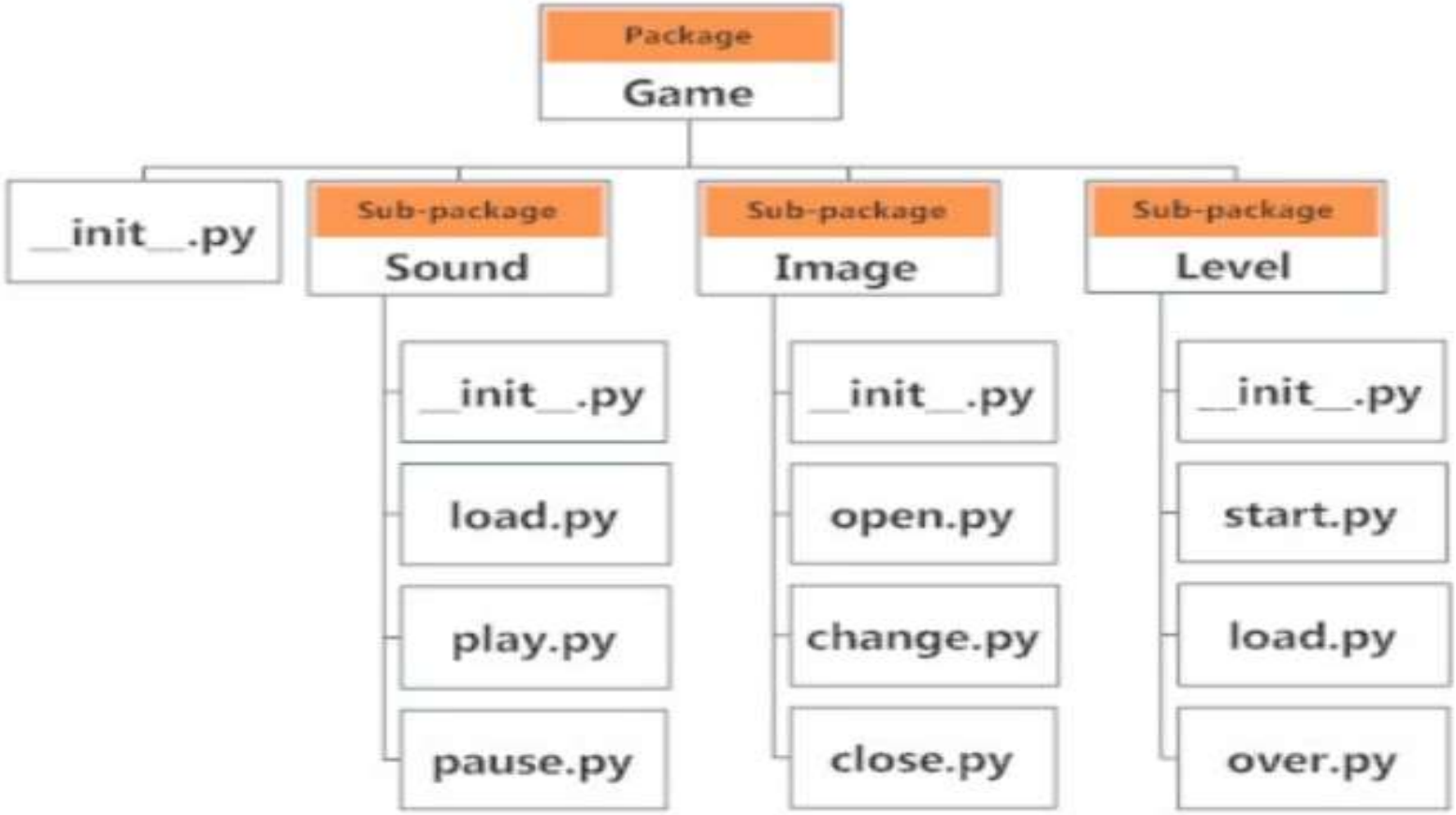
Python Package

- What are packages?
- Importing module from a package

What are packages?

- We don't usually store all of our files in our computer in the same location.
- 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.
- Similar, 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


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

Importing module from a package

- We can import modules from packages using the dot (.) operator.
- For example, if we want to import the start module in the above example, it is done as follows.
  - `import Game.Level.start`
- Now if this module contains a function named select_difficulty(), we must use the full name to reference it.
  - `Game.Level.start.select_difficulty(2)`

- We can also import the module without the package prefix as follows
  - `from Game.Level import start`
- We can now call the function simply as follows.
  - `start.select_difficulty(2)`

- Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows.
  - `from Game.Level.start import select_difficulty`
- Now we can directly call this function.
  - `select_difficulty(2)`

Thank You !