8. Functions

==================================

Python Function

What is a function in Python?

In Python, function is a group of related statements that perform a specific task.

Functions help break our program into smaller and modular chunks. As our 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)
```

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

```
>>> print(greet.__doc__)
        This function greets to
        the person passed into the
        name parameter
```

Types of Functions

Basically, we can divide functions into the following two types:

        Built-in functions - Functions that are built into Python.
        User-defined functions - Functions defined by the users themselves.

Function Argument

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

greet("Monica","Good morning!")
```

Python Arbitrary Arguments

Sometimes, we do not know in advance the number of arguments that will be passed into a function.
Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.
In the function definition we use an asterisk (*) before the parameter name to denote this kind of argument. Here is an example.

```
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("Monica","Luke","Steve","John")
```

Anonymous Function

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?

A lambda function in python has the following syntax.
Syntax of Lambda Function in python

```
lambda arguments: expression
```

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

double = lambda x: x * 2

# Output: 10
print(double(5))
```

is nearly the same as

```
def double(x):
   return x * 2
```

```
# Program to filter out only the even items from a list

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

```
new_list = list(filter(lambda x: (x%2 == 0) , my_list))

# Output: [4, 6, 8, 12]
print(new_list)
```

defining a function

```
def greet_user():
     """Display a simple greeting."""
     print("Hello!")

greet_user()
```

Passing Information to a Function

```
def greet_user(username):
    """Display a simple greeting."""
    print("Hello, " + username.title() + "!")

greet_user('jesse')
```

The variable username in the definition of greet_user() is an example of a parameter, a piece of information the function needs to do its job. The value 'jesse' in greet_user('jesse') is an example of an argument.

Keyword Arguments

A keyword argument is a name-value pair that you pass to a function. You directly associate the name and the value within the argument, so when you pass the argument to the function,

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")

describe_pet(animal_type='hamster', pet_name='harry')
```

Default Values

```
def describe_pet(pet_name, animal_type='dog'):
    """Display information about a pet."""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")
describe_pet(pet_name='willie')
```

return Values

A function doesn’t always have to display its output directly. Instead, it can process some data and then return a value or set of values.

```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = first_name + ' ' + last_name
    return full_name.title()
musician = get_formatted_name('jimi', 'hendrix')
print(musician)

#passing list
def greet_users(names):
    """Print a simple greeting to each user in the list."""
    for name in names:
        msg = "Hello, " + name.title() + "!"
        print(msg)
u usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
```

Passing an arbitrary number of arguments

```
def make_pizza(*toppings):
    """Print the list of toppings that have been requested."""
    print(toppings)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')

def make_pizza(*toppings):
    """Summarize the pizza we are about to make."""
    print("\nMaking a pizza with the following toppings:")
    for topping in toppings:
        print("- " + topping)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

Using Arbitrary Keyword Arguments

Sometimes you’ll want to accept an arbitrary number of arguments, but you won’t know ahead of time what kind of information will be passed to the function. In this case, you can write functions that accept as many key-value pairs as the calling statement provides. One example involves building user profiles: you know you’ll get information about a user, but you’re not sure what kind of information you’ll receive. The function build_profile()

```
def build_profile(first, last, **user_info):
   """Build a dictionary containing everything we know about a user."""
   profile = {}
u     profile['first_name'] = first
   profile['last_name'] = last
v     for key, value in user_info.items():
       profile[key] = value
   return profile
user_profile = build_profile('albert', 'einstein',
                         location='princeton',
                         field='physics')
print(user_profile)
```

Importing Specific Functions
You can also import a specific function from a module. Here's the general syntax for this approach:

```
from module_name import function_name

from module_name import function_0, function_1, function_2

import pizza
```

Using as to Give a Module an Alias

You can also provide an alias for a module name. Giving a module a short alias, like p for pizza, allows you to call the module's functions more quickly. Calling p.make_pizza() is more concise than calling pizza.make_pizza():

```
import pizza as p
p.make_pizza(16, 'pepperoni')
p.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Importing All Functions in a Module
You can tell Python to import every function in a module by using the asterisk (*) operator:

```
from pizza import *

from module_name import *
```

Python Modules
==
What are modules in Python?

Modules refer to a file containing Python statements and definitions.
A file containing Python code, for e.g.: example.py, is called a module and its module name would be example.
We use modules to break down large programs into small manageable and organized files.
Furthermore, modules provide reusability of code.

We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Python import statement

We can import a module using import statement and access the definitions inside
it using the dot operator as described above. Here is an example.

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

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


# import module by renaming it
import math as m
print("The value of pi is", m.pi)

# import only pi from math module
from math import pi
print("The value of pi is", pi)

# import all names from the standard module math
from math import *
print("The value of pi is", pi)

>>> import sys
>>> sys.path
```

The dir() built-in function
We can use the dir() function to find out names that are defined inside a module.
For example, we have defined a function add() in the module example that we had in the beginning.

```
>>> dir(example)
```

Here, we can see a sorted list of names (along with add). All other names that begin with an underscore are default Python attributes associated with the module (we did not define them ourself).

For example, the __name__ attribute contains the name of the module.

```
>>> import example
>>> example.__name__
'example'
```