Unit – V

Secure coding in Python: Interactive Python Scripting, Python Variables, Conditionals, Loops, Functions, External Modules, File operations, Web requests.

Interactive Python Scripting

In Python, there are two options/methods for running code:

- Interactive mode
- Script mode

Interactive Mode

Interactive mode, also known as the REPL provides us with a quick way of running blocks or a single line of Python code. The code executes via the Python shell, which comes with Python installation. Interactive mode is handy when you just want to execute basic Python commands or you are new to Python programming and just want to get your hands dirty with this beautiful language.

To access the Python shell, open the terminal of your operating system and then type "python". Press the enter key and the Python shell will appear. This is the same Python executable you use to execute scripts, which comes installed by default on Mac and Unix-based operating systems.

```
C:\Windows\system32>python
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The >>> indicates that the Python shell is ready to execute and send your commands to the Python interpreter. The result is immediately displayed on the Python shell as soon as the Python interpreter interprets the command.

To run your Python statements, just type them and hit the enter key. You will get the results immediately, unlike in script mode. For example, to print the text "Hello World", we can type the following:

```
>>> print("Hello World")
Hello World
>>>
```

Pros and Cons of Interactive Mode:

The following are the advantages of running your code in interactive mode:

1.Helpful when your script is extremely short and you want immediate results.
2.Faster as you only have to type a command and then press the enter key to get the results.
3.Good for beginners who need to understand Python basics.

The following are the disadvantages of running your code in the interactive mode:

1.Editing the code in interactive mode is hard as you have to move back to the previous commands or else you have to rewrite the whole command again.

2. It's very tedious to run long pieces of code.

Script Mode

If you need to write a long piece of Python code or your Python script spans multiple files, interactive mode is not recommended. Script mode is the way to go in such cases. In script mode, You write your code in a text file then save it with a .py extension which stands for "Python". Note that you can use any text editor for this, including Sublime, Atom, notepad++, etc.

If you are in the standard Python shell, you can click "File" then choose "New" or simply hit "Ctrl + N" on your keyboard to open a blank script in which you can write your code. You can then press "Ctrl + S" to save it.

After writing your code, you can run it by clicking "Run" then "Run Module" or simply press F5.

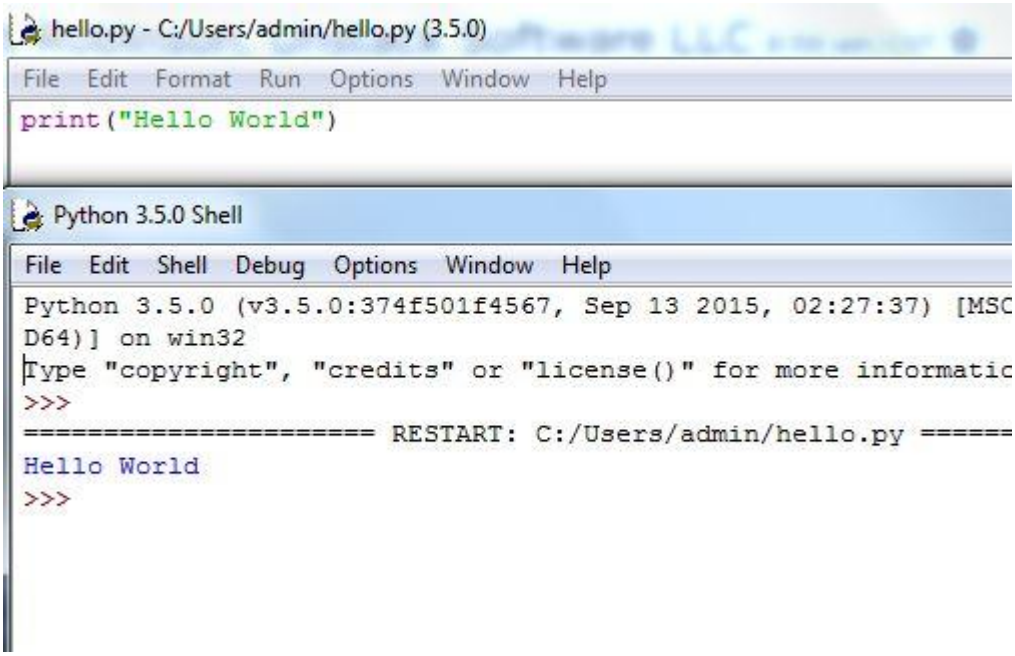
Let us create a new file from the Python shell and give it the name "hello.py". We need to run the "Hello World" program. Add the following code to the file:

```
print("Hello World")
```

Click "Run" then choose "Run Module". This will run the program:

Output:

```
Hello World
```



Pros and Cons of Script Mode

The following are the advantages of running your code in script mode:

1.It is easy to run large pieces of code.
2.Editing your script is easier in script mode.

3.Good for both beginners and experts.

The following are the disadvantages of using the script mode:

1.Can be tedious when you need to run only a single or a few lines of cod.
2.You must create and save a file before executing your code.

Key Differences Between Interactive and Script Mode
Here are the key differences between programming in interactive mode and programming in script mode:
1.In script mode, a file must be created and saved before executing the code to get results. In interactive mode, the result is returned immediately after pressing the enter key.
2.In script mode, you are provided with a direct way of editing your code. This is not possible in interactive mode.

There are two modes through which we can create and run Python scripts: interactive mode and script mode. The interactive mode involves running your codes directly on the Python shell which can be accessed from the terminal of the operating system. In the script mode, you have to create a file, give it a name with a **.py** the extension then runs your code. The interactive mode is suitable when running a few lines of code. The script mode is recommended when you need to create large applications.

Python Variables

Variables
Python variables are the reserved memory locations used to store values with in a Python Program. This means that when you create a variable you reserve some space in the memory.
Creating Variables
1. Python has no command for declaring a variable.
2. A variable is created the moment you first assign a value to it.
Syntax:

```
variable_name = value
```

Example:

```
counter = 1
```

Program:

```
# Program to calculate the area of a rectangle

# Input: length and width of the rectangle
length = float(input("Enter the length of the rectangle: "))
width = float(input("Enter the width of the rectangle: "))

# Calculate the area of the rectangle
area = length * width

# Output: display the calculated area
print("The area of the rectangle is:", area)
```

Output:

```
Enter the length of the rectangle: 5.2
Enter the width of the rectangle: 3.8
The area of the rectangle is: 19.76
```

Rules for Python variables

1. A Python variable name must start with a letter or the underscore character.
2. A Python variable name cannot start with a number.
3. A Python variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
4. Variable in Python names are case-sensitive (name, Name, and NAME are three different variables).
5. The reserved words(keywords) in Python cannot be used to name the variable in Python.

Types of variables in Python

Python has two types of variables: global variables and local variables.
To utilize the variable in other parts of your program or module, you need to declare it as a global one. It is common practice in Python to use local variables when creating new variables.

Local variables in Python are the ones that are defined and declared inside a function. We can not call this variable outside the function.

Example:

```
def sum(x,y):
    sum = x + y
    return sum
print(sum(5, 10))
```

```
15
```

Global variables in Python are the ones that are defined and declared outside a function, and we need to use them inside a function.

Example:

```
x = 5
y = 10
def sum():
    sum = x + y
    return sum
print(sum())
```

```
15
```

Conditionals

1) if

Syntax:

if condition : statement

Or

if condition :
statement-1
statement-2
statement-3

If condition is true then statements will be executed.

Example:

```
1) name=input("Enter Name:")
2) if name=="durga" :
3)     print("Hello Durga Good Morning")
4) print("How are you!!!")
```

Output:

```
D:\Python_classes>py test.py
Enter Name:durga
Hello Durga Good Morning
How are you!!!
```

```
D:\Python_classes>py test.py
Enter Name:Ravi
How are you!!!
```

2) if - else
Syntax:
if condition :
Action-1
else :
Action-2

if condition is true then Action-1 will be executed otherwise Action-2 will be executed.

Example:

```
1) name=input("Enter Name:")
2) if name=="durga" :
3)     print("Hello Durga Good Morning")
4) else:
5)     print("Hello Guest Good Moring")
6) print("How are you!!!")
```

Output:

```
D:\Python_classes>py test.py
Enter Name:durga
Hello Durga Good Morning
How are you!!!
```

```
D:\Python_classes>py test.py
Enter Name:Ravi
Hello Guest Good Moring
How are you!!!
```

3) if – elif - else

Syntax:

```
if condition1 :
   Action-1
elif condition2:
   Action-2
elif condition3:
   Action-3
elif condition4:
   Action-4
   …
else :
   Default Action
```

Based on the condition the corresponding action will be executed

Example:

```
1) name=input("Enter Name:")
2) if name=="durga":
3)    print("Hello Durga Good Morning")
4) elif name=="ravi":
5)    print("Hello Ravi Good Morning")
6) elif name=="raju" :
7)    print("Hello Raju Good Morning")
8) else:
9)    print("How are you!!!")
```

Output:

```
D:\Python_classes>py test.py
Enter Name:durga
Hello Durga Good Morning
```

```
D:\Python_classes>py test.py
Enter Name:raju
Hello Raju Good Morning
```

```
D:\Python_classes>py test.py
Enter Name:rajesh
How are you!!!
```

Note:

1. else part is always optional
   Hence the following are various possible syntaxes.
   1. if
   2. if - else
   3. if-elif-else
   4. if-elif
2. There is no switch statement in Python

Program1:

#Write a Python program that calculates a student's grade based on their score.

```
# Get the student's score
score = float(input("Enter the student's score: "))

# Determine the grade based on the score
if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
elif score >= 60:
    grade = 'D'
else:
    grade = 'F'

# Display the grade
print(f"The student's grade is: {grade}")
```

Output:

```
Enter the student's score: 85
The student's grade is: B
```

Program2:

Write a program to find biggest of given 3 numbers from the command prompt?

```
1) n1=int(input("Enter First Number:"))
2) n2=int(input("Enter Second Number:"))
3) n3=int(input("Enter Third Number:"))
4) if n1>n2 and n1>n3:
5)     print("Biggest Number is:",n1)
6) elif n2>n3:
7)     print("Biggest Number is:",n2)
8) else :
9)     print("Biggest Number is:",n3)
```

Output:

```
Enter First Number:10
Enter Second Number:20
Enter Third Number:30
Biggest Number is: 30
```

Loops

Iterative Statements

• If we want to execute a group of statements multiple times then we should go for Iterative statements.

• Python supports 2 types of iterative statements.

1. for loop

2. while loop

for loop:
If we want to execute some action for every element present in some sequence (it may be string or collection)then we should go for for loop.
Syntax:

for x in sequence :

body

where sequence can be string or any collection.
Body will be executed for every element present in the sequence.

Example1: To display numbers from 1 to 10

```
for x in range(1,11):
        print(x)
```

Example2: To display odd numbers from 0 to 20

```
for x in range(21):
        if(x%2!=0):
                print(x)
```

Example3: To display numbers from 10 to 1 in descending order

```
for i in range(10,0,-1):
        print(i)
```

Example4: To print characters present in the given string

```
s=input("Enter a String\n")
for ch in s:
    print(ch)
```

Output:

```
Enter a String
ADITYA
A
D
I
T
Y
A
```

Example5: To print characters present in string index wise

```
s=input("Enter some String:\n")
i=0
for ch in s:
    print("The character present at",i,"index is:",ch)
    i=i+1
```

Output:

```
Enter some String:
CSE
The character present at 0 index is: C
The character present at 1 index is: S
The character present at 2 index is: E
```

while loop:
If we want to execute a group of statements iteratively until some condition false, then we should go for while loop.

Syntax:
while condition:
    body

Example1: To print numbers from 1 to 10 by using while loop

```
x=1
while x<=10:
    print(x)
    x=x+1
```

Example2: To display the sum of first n numbers

```
1) n=int(input("Enter number:"))
2) sum=0
3) i=1
4) while i<=n:
5)     sum=sum+i
6)     i=i+1
7) print("The sum of first",n,"numbers is :",sum)
```

Functions

• If a group of statements is repeatedly required then it is not recommended to write these statements every time separately. We have to define these statements as a single unit and we can call that unit any number of times based on our requirement without rewriting. This unit is nothing but function.

• The main advantage of functions is code Reusability.

• **Note:** In other languages functions are known as methods, procedures, subroutines Etc

Python supports 2 types of functions

1. Built in Functions
2. User Defined Functions

1. Built in Functions:
The functions which are coming along with Python software automatically, are called built in functions or pre defined functions
Ex:
id()
type()
input()
eval()

etc..

2. User Defined Functions:
The functions which are developed by programmer explicitly according to business requirements are called user defined functions.
Syntax :

```
def function_name(parameters):

""" doc string"""
----
-----
return value
```

Note:

While creating functions we can use 2 keywords

1. def (mandatory)
2. return (optional)

Example1:

```
def display():
        print("Hello Good Morning")
display()
display()
display()
```

Example2:

```
def display():
        print("Hello Good Morning")
def wish():
        display()
wish()
display()
wish()
```

Parameters:
Parameters are inputs to the function. If a function contains parameters, then at the time of calling, compulsory we should provide values otherwise, otherwise we will get error.

Example3:
#Write a function to take name of the student as input and print wish message by name.

```
1. def wish(name):
2.     print("Hello",name," Good Morning")
3. wish("Durga")
4. wish("Ravi")
```

Output:

```
D:\Python_classes>py test.py
Hello Durga  Good Morning
Hello Ravi  Good Morning
```

Example4:
#Write a function to take number as input and print its square value.

```
1. def squareIt(number):
2.     print("The Square of",number,"is", number*number)
3. squareIt(4)
4. squareIt(5)
```

Output:

```
D:\Python_classes>py test.py
The Square of 4 is 16
The Square of 5 is 25
```

Return Statement:
Function can take input values as parameters and executes business logic, and returns output to the caller with return statement.
Example5: Write a function to accept 2 numbers as input and return sum.

```
1. def add(x,y):
2.     return x+y
3. result=add(10,20)
4. print("The sum is",result)
5. print("The sum is",add(100,200))
```

Output:

```
D:\Python_classes>py test.py
The sum is 30
The sum is 300
```

Example6:
If we are not writing return statement then default return value is None

```
1. def f1():
2.     print("Hello")
3. f1()
4. print(f1())
```

Output:

```
Hello
Hello
None
```

Example 7 : Write a function to find factorial of given number?

```
1)  def fact(num):
2)      result=1
3)      while num>=1:
4)          result=result*num
5)          num=num-1
6)      return result
7)  for i in range(1,5):
8)      print("The Factorial of",i,"is :",fact(i))
```

Output:

```
D:\Python_classes>py test.py
The Factorial of 1 is : 1
The Factorial of 2 is : 2
The Factorial of 3 is : 6
The Factorial of 4 is : 24
```

Types of arguments:

```
def f1(a,b):
------
------

------
f1(10,20)
```

a, b are formal arguments where as 10,20 are actual arguments

There are 4 types are actual arguments are allowed in Python.

1. positional arguments
2. keyword arguments
3. default arguments
4. Variable length arguments

1. positional arguments:

• These are the arguments passed to function in correct positional order.

```
def sub(a,b):
        print(a-b)
sub(100,200)
sub(200,100)
```

• The number of arguments and position of arguments must be matched. If we change the order then result may be changed.

• If we change the number of arguments then we will get error.

2. keyword arguments:
We can pass argument values by keyword i.e by parameter name.
Example:

```
1. def wish(name,msg):
2.     print("Hello",name,msg)
3. wish(name="Durga",msg="Good Morning")
4. wish(msg="Good Morning",name="Durga")
```

Output:

```
Hello Durga Good Morning
Hello Durga Good Morning
```

Note: We can use both positional and keyword arguments simultaneously. But first we have to take positional arguments and then keyword arguments, otherwise we will get Syntax error.

```
def wish(name,msg):
    print("Hello",name,msg)
wish("Durga","GoodMorning") → Valid
wish("Durga",msg="GoodMorning") → Valid
wish(name="Durga","GoodMorning") → Invalid
SyntaxError: positional argument follows keyword argument
```

3. Default Arguments:
Sometimes we can provide default values for our positional arguments.
Example:

```
1) def wish(name="Guest"):
2)     print("Hello",name,"Good Morning")
3) wish("Durga")
4) wish()
```

Output:

```
Hello Durga Good Morning
Hello Guest Good Morning
```

Note: After default arguments we should not take non default arguments

```
1) def wish(name="Guest",msg="Good Morning"): → Valid
2) def wish(name,msg="Good Morning"): → Valid
3) def wish(name="Guest",msg): → Invalid
```

```
SyntaxError: non-default argument follows default argument
```

4. Variable length arguments:

• Sometimes we can pass variable number of arguments to our function,such type of arguments are called variable length arguments.

• We can declare a variable length argument with * symbol as follows

def f1(*n):

• We can call this function by passing any number of arguments including zero number.

• Internally all these values represented in the form of tuple.

Example:

```
1)  def sum(*n):
2)      total=0
3)      for n1 in n:
4)          total=total+n1
5)      print("The Sum=",total)
6)
7)  sum()
8)  sum(10)
9)  sum(10,20)
10) sum(10,20,30,40)
```

Output:

```
The Sum= 0
The Sum= 10
The Sum= 30
The Sum= 100
```

External Modules

External Modules There are many other libraries that have been built by developers outside of the core Python team, to add additional functionality to the language. These modules don't come as part of the Python language, but can be added in. We call these **external modules.**

In order to use an external module, you must first install it on your machine. This means you'll need to download the files from the internet to your computer, then integrate them with the main python library, so that the language knows where the module is located.

pip

It is usually possible to install modules manually, but this process can be a major pain. Luckily, python also gives us a streamlined approach for installing modules- the **pip module!** This feature can locate modules that are indexed in the Python Package Index (a list of commonly-used modules), download them, and attempt to install them.

Traditionally, we don't run pip from our normal editor- instead, you'll need to run it from the terminal. This is a command interface that lets you make changes directly to your computer. On Mac and Linux machines, you can find the **terminal** by searching your applications for the built-in app Terminal. On Windows, search for the built-in application Powershell.

Syntax:

pip install module_name

pip install numpy

```python
import numpy as np

def main():
    # Create a simple 1-dimensional array
    arr = np.array([1, 2, 3, 4, 5])
    print("Array:", arr)

    # Perform operations on the array
    print("Sum:", np.sum(arr))
    print("Mean:", np.mean(arr))
    print("Max:", np.max(arr))
    print("Min:", np.min(arr))

    # Create a 2-dimensional array
    arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
    print("2D Array:")
    print(arr2d)

    # Perform operations on the 2D array
    print("Sum (axis 0):", np.sum(arr2d, axis=0))  # Sum along axis 0 (columns)
    print("Sum (axis 1):", np.sum(arr2d, axis=1))  # Sum along axis 1 (rows)

if __name__ == "__main__":
    main()
```

```
Array: [1 2 3 4 5]
Sum: 15
Mean: 3.0
Max: 5
Min: 1
2D Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Sum (axis 0): [12 15 18]
Sum (axis 1): [ 6 15 24]
```

File operations

Files

• As the part of programming requirement, we have to store our data permanently for future purpose. For this requirement we should go for files.
• Files are very common permanent storage areas to store our data.

Types of Files:
There are 2 types of files
1. Text Files:

Usually we can use text files to store character data
eg: abc.txt
2. Binary Files:

Usually we can use binary files to store binary data like images,video files, audio

files etc…

1.Opening a File
Before performing any operation (like read or write) on the file, first we have to open that file. For this we should use Python's inbuilt function open()
• But at the time of open, we have to specify mode, which represents the purpose of opening file.

f = open(filename, mode)

The allowed modes in Python are
1. r → open an existing file for read operation. The file pointer is positioned at the beginning of the file. If the specified file does not exist then we will get FileNotFoundError. This is default mode.
2. w → open an existing file for write operation. If the file already contains some data then it will be overridden. If the specified file is not already available then this mode will create that file.
3. a → open an existing file for append operation. It won't override existing data. If the specified file is not already available then this mode will create a new file.
4. r+ → To read and write data into the file. The previous data in the file will not be deleted. The file pointer is placed at the beginning of the file.
5. w+ → To write and read data. It will override existing data.
6. a+ → To append and read data from the file. It wont override existing data.
7. x → To open a file in exclusive creation mode for write operation. If the file already exists then we will get **FileExistsError.**

Note: All the above modes are applicable for text files. If the above modes suffixed with 'b' then these represents for binary files.

Eg: rb, wb, ab, r+b, w+b, a+b, xb

f = open("abc.txt","w")
We are opening abc.txt file for writing data.

2.Closing a File:
After completing our operations on the file, it is highly recommended to close the file. For this we have to use close() function.

f.close()

3. Various properties of File Object
Once we opened a file and we got file object, we can get various details related to that file by using its properties.
• name → Name of opened file
• mode → Mode in which the file is opened
• closed → Returns boolean value indicates that file is closed or not
• readable()→ Returns boolean value indicates that whether file is readable or not
• writable()→ Returns boolean value indicates that whether file is writable or not.

Example

```
f=open("abc.txt",'w')
print("File Name: ",f.name)
print("File Mode: ",f.mode)
print("Is File Readable:  ",f.readable())
print("Is File Writable:  ",f.writable())
print("Is File Closed : ",f.closed)
f.close()
print("Is File Closed : ",f.closed)
```

Output:

```
D:\Python_classes>py test.py
File Name:  abc.txt
File Mode:  w
Is File Readable:   False
Is File Writable:   True
Is File Closed :  False
Is File Closed :  True
```

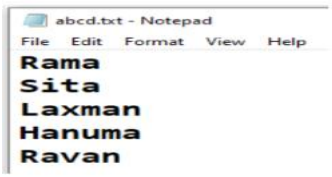
4. Writing data to text files

We can write character data to the text files by using the following 2 methods.
1)write(str)
2)writelines(list of lines)

1) write(str)

```
f=open("abcd.txt",'w')
f.write("Rama\n")
f.write("Sita\n")
f.write("Laxman\n")
f.write("Hanuma\n")
f.write("Ravan\n")
print("Data written to the file successfully")
f.close()
```

Output:



Note: In the above program, data present in the file will be overridden every time if we run the program. Instead of overriding if we want append operation then we should open the file as follows.

f = open("abcd.txt","a")

2)writelines(list)

```
f=open("abcd.txt",'w')
list=["sunny\n","bunny\n","vinny\n","chinny"]
f.writelines(list)
print("List of lines written to the file successfully")
f.close()
```

Output:

Note: while writing data by using write() methods, compulsory we have to provide line seperator (\n), otherwise total data should be written to a single line.

5. Reading Character Data from text files

We can read character data from text file by using the following read methods.

- read() → To read total data from the file
- read(n) → To read 'n' characters from the file
- readline()→ To read only one line
- readlines()→ To read all lines into a list

Example: To read all lines into list

```
f=open("abcd.txt","r")
print(f.read(3))
print(f.readline())
print(f.read(4))
print("Remaining data")
print(f.read())
```

Output:

```
sun
ny

bunn
Remaining data
y
vinny
chinny
```

Web Requests

Web Requests in Python using requests Library

1. Importing the requests Library
Use import requests to import the requests library in Python.

2. Making a GET Request
Use requests.get(url) to make a GET request to a specified URL.
Returns a response object containing the server's response.

3. Checking the Response Status
Access the response status code using response.status_code.
Common status codes: 200 (OK), 404 (Not Found), 500 (Server Error).

4. Accessing Response Content
Use response.text to get the response content as text.

5. Sending Query Parameters
Pass parameters using params argument in requests.get(url, params=params).
Parameters added to the URL for a GET request.

6. Making a POST Request
Use requests.post(url, data=data) to make a POST request with data.
data contains the payload to be sent with the request.

7. Handling JSON Responses
Use response.json() to parse JSON data in the response to a Python dictionary.

These fundamental concepts cover making web requests, handling responses, sending parameters, making POST requests, and dealing with JSON data. The requests library is versatile and widely used for interacting with web APIs and fetching data from the internet in Python.

Example:
1. Making a GET Request

```
import requests

url = 'https://jsonplaceholder.typicode.com/posts/1'
response = requests.get(url)

print("Response Content:")
print(response.text if response.status_code == 200 else "Request failed")
```

Output:
(JSON - like format)

```
Response Content:
{ "userId": 1, "id": 1, "title": "sunt aut facere repellat provident occaecati...", "body": "quia et suscipit\nsuscipit... }
```

2. Sending Query Parameters

```
import requests

url = 'https://jsonplaceholder.typicode.com/posts'
params = {'userId': 1}
response = requests.get(url, params=params)

print("Response Content:")
print(response.text if response.status_code == 200 else "Request failed")
```

Output:
(JSON - like format)

```
Response Content:
[{ "userId": 1, "id": 1, "title": "sunt aut facere repellat provident occaecati...", "body": "quia et suscipit\nsuscipit... }, ...]
```

3. Making a POST Request

```
import requests

url = 'https://jsonplaceholder.typicode.com/posts'
data = {'title': 'foo', 'body': 'bar', 'userId': 1}
response = requests.post(url, data=data)

print("Response Content:")
print(response.json() if response.status_code == 201 else "Request failed")
```

Output:
(JSON - like format)

```
Response Content:
{ "title": "foo", "body": "bar", "userId": "1", "id": 101 }
```

4. Handling JSON Responses

```
import requests

url = 'https://jsonplaceholder.typicode.com/posts/1'
response = requests.get(url)

if response.status_code == 200:
    post_data = response.json()
    print("Post Title:", post_data['title'])
    print("Post Body:", post_data['body'])
else:
    print("Request failed")
```

Output:
(css -like format)

```
Post Title: sunt aut facere repellat provident occaecati...
Post Body: quia et suscipit\nsuscipit...
```

Secure Coding Standards for Python: Best Practices

1. Input Validation

User input is often a significant source of security risks. Input validation is the process of verifying that the user input meets the expected criteria and is safe to use in the application.

For example, when a user enters a credit card number, the input should only contain digits and no special characters. To validate the input, developers can use built-in functions such as isdigit() or regular expressions to ensure that the input meets the expected criteria.

```python
import re

def validate_input(input_string):
    """
    Function to validate input using regular expressions.
    """
    pattern = r"^[0-9]+$"
    if re.match(pattern, input_string):
        return True
    else:
        return False
```

2. Avoid Using Unsafe Functions

Python has several functions that can be vulnerable to security issues if not used carefully. Functions such as exec(), eval(), and pickle can allow attackers to execute malicious code. Developers should avoid using these functions or use them with caution by restricting input parameters and using them only when necessary.

For example, instead of using eval() function to convert a string to an integer, developers should use the int() function.

```python
# Instead of using eval function
x = eval('10')

# Use int function
x = int('10')
```

3. Use Cryptography Libraries

Cryptography libraries such as cryptography and pycryptodome provide a secure way to perform encryption and decryption operations. Use these libraries instead of creating custom encryption methods, which may be prone to vulnerabilities.

For example, to encrypt a password, use the cryptography library as follows:

```
from cryptography.fernet import Fernet

def encrypt_password(password):
    """
    Function to encrypt password using cryptography library.
    """
    key = Fernet.generate_key()
    f = Fernet(key)
    encrypted_password = f.encrypt(password.encode('utf-8'))
    return encrypted_password

password = "mypassword"
encrypted_password = encrypt_password(password)
```

4. Follow the Principle of Least Privilege

The principle of least privilege is a security best practice that restricts users or processes to the minimum level of access necessary to perform their functions. Developers should follow this principle when writing code to minimize the impact of security breaches.

For example, if an application requires read-only access to a database, it should use a database account with read-only permissions instead of an account with full permissions. This reduces the risk of an attacker exploiting the application to modify or delete data.

5. Keep Libraries and Frameworks Updated

Libraries and frameworks can contain security vulnerabilities that can be exploited by attackers. Developers should keep their libraries and frameworks updated to the latest version to avoid potential security issues.

For example, if the application uses a third-party library, such as Requests , which has a security vulnerability, the developer should update to the latest version of the library that addresses the vulnerability.

6. Use a Static Code Analyzer

A static code analyzer is a tool that can identify potential security vulnerabilities in the code before it is executed. Use tools such as bandit , Pylint , and Pyflakes to detect security issues in the code and fix them before deployment.

For example, bandit is a popular static code analyzer that examines Python code for potential security vulnerabilities. It can detect issues such as hard-coded passwords, SQL injection, and use of unsafe functions.

7. Use Secure Coding Practices for Web Applications

Web applications are vulnerable to several security risks such as cross-site scripting, SQL injection, and command injection. Developers should follow secure coding practices such as input validation, output encoding, and parameterized queries to ensure that web applications are secure.

For example, when writing SQL queries, use parameterized queries instead of concatenating user input with the query. Parameterized queries prevent SQL injection attacks by treating user input as data rather than executable code.

```
# Instead of this
query = "SELECT * FROM users WHERE username = '" + username + "';"

# Use parameterized query
query = "SELECT * FROM users WHERE username = %s;"
cursor.execute(query, (username,))
```