


```

def hello(name):
    print('Hello ' + name)

hello('Alice')
hello('Bob')

```

2. None

```

>>> spam = print('Hello!')
Hello!
>>> None == spam
True

```

In Python there is a value called None, which represents the absence of a value. None is the only value of the None Type data type.

3. not in

We can determine whether a value is or isn't in a list with the in and not in operators.

in and **not in** are used in expressions and connect two values: a value to look for in a list and the list where it may be found and these expressions will evaluate to a Boolean value

```

>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
False
>>> 'howdy' not in spam
False
>>> 'cat' not in spam
True

```

4. del The del statement will delete values at an **index** in a list. All of the values in the list after the deleted value will be moved up one index. Ex:

```

>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
>>> del spam[2]

```

2a)

What is the difference between break and continue statement? Explain with programs(2+2)

[4]

CO1

L2

break

1. There is a shortcut to getting the program execution to break out of a while loop's clause early
 2. If the execution reaches a break statement, it immediately exits the while loop's clause.

continue

1. Like break statements, continue statements are used inside loops
 2. When the program execution reaches a continue statement, the program execution immediately jumps back to the start of the loop and reevaluates the loop's condition.

```

1 while True:
2     print('Please type your name.')
3     name = input()
4     if name == 'your name':
5         break
6     print('Thank you!')

```

```

while True:
    print('Who are you?')
    name = input()
    if name != 'Joe':
        continue
    print('Hello, Joe. What is the password? (It is a fish.)')
    password = input()
    if password == 'swordfish':
        break
    print('Access granted.')

```

2b) Explain the concept of Local scope and global scope of the variables in Python with example code snippets. (3*2=6)

[6] CO1 L2

Explanation -3

Example-3

- Parameters and variables that are assigned in a called function are said to exist in that function's localscope.
- Variables that are assigned outside all functions are said to exist in the global scope.
- A variable that exists in a local scope is called a local variable, while a variable that exists in the globalscope is called a global variable.
- A variable must be one or the other; it cannot be both local and global.
- When a scope is destroyed, all the values stored in the scope's variables are forgotten.
- There is only one global scope, and it is created when your program begins. When your program terminates, the global scope is destroyed, and all its variables are forgotten.
- A local scope is created whenever a function is called. Any variables assigned in this function exist within the local scope. When the function returns, the local scope is destroyed, and these variables are forgotten.
- Scopes matter for several reasons:
 1. Code in the global scope cannot use any local variables.
 2. However, a local scope can access global variables.
 3. Code in a function's local scope cannot use variables in any other local scope.
 4. We can use the same name for different variables if they are in different scopes. That is, there can be a local variable named spam and a global variable also named spam.

```

def spam():
1     eggs = 'spam local'
2     print(eggs)    # prints 'spam local'

def bacon():
3     eggs = 'bacon local'
4     print(eggs)    # prints 'bacon local'
5     spam()
6     print(eggs)    # prints 'bacon local'

7 eggs = 'global'
8 bacon()
9 print(eggs)        # prints 'global'

```

- 1 A variable named eggs that exists in a local scope when spam() is called.
- 2 A variable named eggs that exists in a local scope when bacon() is called.
- 3 A variable named eggs that exists in the global scope.

3(a)	<p>Explain the difference between function and methods with examples(2+2) Explanation and Example (2+2)</p> <ul style="list-style-type: none"> ➤ A function is like a mini-program within a program. <pre style="margin-left: 40px;"> 1 def hello(): 2 print('Howdy!') 3 print('Howdy!!!') 4 print('Hello there.')</pre> <pre style="margin-left: 40px;"> 1 hello() 2 hello() 3 hello()</pre> <ul style="list-style-type: none"> ➤ Example: ➤ The first line is a def statement 1, which defines a function named hello(). ➤ The code in the block that follows the def statement 2 is the body of the function. This code is executed when the function is called, not when the function is first defined. ➤ The hello() lines after the function 3 are function calls. ➤ In code, a function call is just the function's name followed by parentheses, possibly with some number of arguments in between the parentheses. <p>Methods</p> <ul style="list-style-type: none"> • A method is the same thing as a function, except it is “called on” a value. • Each data type has its own set of methods. • The list data type, has several useful methods for finding, adding, removing, and otherwise manipulating values in a list. • List.append() 	[4]	CO1	L2
3(b)	<p>Write a Python Program to check if a given number is Fibonacci number or not? (4+2=6)</p> <p>Program code=4 Output=2</p> <p>Input: 13 Output: True</p> <p>Input: 25 Output: False</p> <p>Program:</p> <pre> def fib(n): c=0 a=1 b=1 if n==0 or n==1: print("True") else: while c<n: c=a+b b=a a=c if c==n: print("True") else: print("False")</pre>	[6]	CO1	L3


```
>>> spam[0:4] ['cat', 'bat', 'rat', 'elephant']
>>> spam[1:3] ['bat', 'rat']
>>> spam[0:-1] ['cat', 'bat', 'rat']
```

6(b)

What are exceptions in Python? How it can be handled in Python? Exception Handling (4+2)

Explanation:4

Example:2

Exception Handling

- If we don't want to crash the program due to errors instead we want the program to detect errors, handle them, and then continue to run.
- For example,

Program

Output

```
def spam(divideBy):
    return 42 / divideBy
```

```
print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

```
21.0
3.5
Traceback (most recent call last):
  File "C:/zeroDivide.py", line 6, in <module>
    print(spam(0))
  File "C:/zeroDivide.py", line 2, in spam
    return 42 / divideBy
ZeroDivisionError: division by zero
```

- A ZeroDivisionError happens whenever we try to divide a number by zero. From the line number given in the error message, we know that the return statement in spam() is causing an error.
- Errors can be handled with try and except statements.
- The code that could potentially have an error is put in a try clause. The program execution moves to the start of a following except clause if an error happens.
- We can put the previous divide-by-zero code in a try clause and have an except clause contain code to handle what happens when this error occurs.

Program

Output

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')
```

```
print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

```
21.0
3.5
Error: Invalid argument.
None
42.0
```

any errors that occur in function calls in a try block will also be caught.

[6] CO1 L2