PCAP-31-03

Exam A

QUESTION 1
What will be the value of the i variable when the while loop finishes its execution?

```
i=0
while i  ! =0:
    i=i-1
else:
    i=i+1
```

A. 1
B. 0
C. 2
D. the variable becomes unavailable

Correct Answer: A
Explanation

Explanation/Reference:
When entering the `while` loop, since `i` is set to 0 at the begining, the condition `i != 0` returns False.
The suite in `while` clause is NOT run.
Then `else` clause is run for one time since condition returns `False`. This adds 1 to `i.`
Therefore `i` will have a value of 1 after the `while` statement.

Code for testing:

```
i=0
while i !=0:
~~~~i=i-1
else:
~~~~i=i+1
print(i)
```

QUESTION 2
The operator that can be able to perform bitwise shifts is coded as (Choose two.)

A. `-`
B. `++`
C. `<<`
D. `>>`

Correct Answer: CD
Explanation

Explanation/Reference:

QUESTION 3
What will the value of the i variable be when the following loop finishes its execution?

```
for i in range (10):
    pass
```

A. 10
B. the variable becomes unavailable
C. 11
D. 9

Correct Answer: D
Explanation

Explanation/Reference:
Although pass does nothing, the number sequence will continue to iterate until the last number i.e. 9 is reached.

Code for testing:

```
for i in range(10):
~~~~pass
print(i)
```

QUESTION 4
The following expression

```
1+-2
```

is:

A. equal to 1
B. invalid
C. equal to 2
D. equal to -1

Correct Answer: D
Explanation

Explanation/Reference:

QUESTION 5
A compiler is a program designed to (Choose two.)

A. rearrange the source code to make it clearer
B. check the source code in order to see if syntax correct
C. execute the source code
D. translate the source code into machine code

Correct Answer: BD
Explanation

Explanation/Reference:

QUESTION 6

What is the output of the following piece of code?

```
a= ‘ant’
b= “bat”
c= ‘camel’
print (a, b, c, sep= ‘””’)
```

A. ant'bat'camel
B. ant"bat"camel
C. antbatcamel
D. print (a, b, c, sep="")

Correct Answer: B
Explanation

Explanation/Reference:
Code for testing:

```
a = 'ant'
b = "bat"
c = 'camel'
print (a, b, c, sep= '"') # ant"bat"camel
```

QUESTION 7
What is the expected output of the following snippet?

```
i=5
while i>0:
    i=i //2
    if i % 2=0:
        break
else:
    i+=1
print (i)
```

A. the code is erroneous
B. 3
C. 7
D. 15

Correct Answer: A

Explanation

Explanation/Reference:
The condition in the `if` header is wrong, comparison operator "==" should be used instead of the assignment operator "="

Code for testing (the operator has been replaced).

```
i=5
while i>0:
~~~~i=i//2
~~~~if i%2==0:
~~~~~~~~break
else:
~~~~i+=1
print(i)
```

Remarks:
After changing "=" to "==", the result will be 2 since:
- In the first loop, 5 // 2 is the integer part of 2.5 i.e. i is therefore assigned with 2
- 2 % 2 is 0 and will break out of the while loop (skipping else clause of while)
- Since i is 2, the printout will be 2.

QUESTION 8
How many lines does the following snippet output?

```
for i in range (1, 3):
    print (“*”, end= “”)
else:
    print (“*”)
```

A. three
B. one
C. two
D. four

Correct Answer: B
Explanation

Explanation/Reference:
The number sequence for 1,3 is 1,2 (excluding 3).
For each number iterated, a "*" is printed. Therefore two "*" are printed during iteration.
Finally, after all elements are iterated, else clause is run to print one more "*"
Therefore a total of three "*" are printed.
However, since the first two print() has end="", therefore no newline character is included and therefore all "*" are printed in the same line.

Code for testing:

```
for i in range(1,3):
~~~~print("*", end="")
else:
~~~~print("*")
```

QUESTION 9
Which of the following literals reflect the value given as 34.23? (Choose two.)

A. .3423e2
B. 3423e-2
C. .3423e-2
D. 3423e2

Correct Answer: AB
Explanation

Explanation/Reference:
1e-2 is 0.01

Therefore 34.23 can also represented by any one of the followings:
```
3423e-2
342.3e-1
34.23e0
3.423e1
0.3423e2
```

QUESTION 10
What is the expected output of the following snippet?

```
a=2
if a>0:
    a+=1
    else:
        a-=1
print(a)
```

A. 3
B. 1
C. 2
D. the code is erroneous

Correct Answer: D
Explanation

Explanation/Reference:

Within the same `if` compound statement,. since the `if` header should have the same identation as the `else` header, there is a syntax error.

Code for testing:

```
a=2
if a>0:
~~~~a+=1
~~~~else:
~~~~~~~~a-=1
print(a)
```

Remarks:
However, if you fixed the above as follows:

```
a=2
if a>0:
~~~~a+=1
else:
~~~~a-=1
print(a)
```

The result will be 3 since:
- Since a is set to 2, the condition of `if` returns `True`.
- `a+=1` is run to add 1 to a. Hence a will become 3.
- `else` clause in if statement will not run if confition is `True`.

QUESTION 11
Assuming that the following snippet has been successfully executed, which of the equations are True? (Choose two.)

```
a= [1]
b=a
a[0] = 0
```

A. len(a) == len(b)
B. b[0] +1 ==a[0]
C. a[0] == b[0]
D. a[0] + 1 == b[0]

Correct Answer: AC
Explanation

Explanation/Reference:
Due to the assignment "b = a", "a" and "b" refers to the same List object.

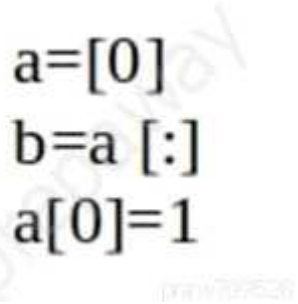
Code for testing:

```
a = [1]
b = a
a[0] = 0

print(len(a) == len(b))   # True
print(b[0] + 1 == a[0])   # False
print(a[0] == b[0])       # True
```

```
print(a[0] + 1 == b[0])   # False
```

QUESTION 12
Assuming that the following snippet has been successfully executed, which of the equations are False? (Choose two.)



A. len(a)== len(b)
B. a[0]-1 == b[0]
C. a[0] == b[0]
D. b[0] - 1 == a[0]

Correct Answer: AB
Explanation

Explanation/Reference:
Due to the assignment "b = a[:]" is the same creating a copy of "a" and assign the new List object to "b". They are two different List objects.

Code for testing:

```
a = [0]
b = a[:]
a[0] = 1

print(a)                # [1]
print(b)                # [0]
print(len(a) == len(b))   # True
print(a[0] - 1 == b[0])   # True
print(a[0] == b[0])       # False
print(b[0] - 1 == a[0])   # False
```

QUESTION 13
Which of the following statements are true? (Choose two.)

A. Python strings are actually lists
B. Python strings can be concatenated
C. Python strings can be sliced like lists
D. Python strings are mutable

Correct Answer: BC
Explanation

Explanation/Reference:
Instead of List, Python string is more similar to Tuple since it is immutable.
However, Python string is NOT actually Tuple or List since they are of different type and have different methods.

QUESTION 14
Which of the following sentences are true? (Choose two.)

A. Lists may not be stored inside tuples
B. Tuples may be stored inside lists
C. Tuples may not be stored inside tuples
D. Lists may be stored inside lists

Correct Answer: BD
Explanation

Explanation/Reference:
List / Tuple can be stored in each other. Both can also store other container types.
However, a Set can only store Tuple.

QUESTION 15
Assuming that String is six or more letters long, the following slice

```
string [1:-2]
```

is shorter than the original string by:

A. four chars
B. three chars
C. one char
D. two chars

Correct Answer: B
Explanation

Explanation/Reference:
You should remember that end is exclusive and then you can draw something like the following to help you to choose the answer.

```
 a  b  c  d  e  f
 0  1  2  3  4  5
-6 -5 -4 -3 -2 -1
   |<------>|
```

Code for testing:

```
string1 = "abcdef"
sliced1 = string1[1:-2]
print(len(string1)," : ",len(sliced1))   # 6 : 3
string2 = "1234567"
sliced2 = string2[1:-2]
print(len(string2)," : ",len(sliced2))   # 7 : 4
```

QUESTION 16
What is the expected output of the following snippet?

```
lst = [1,2,3,4]
lst = lst [-3:-2]
lst= lst[-1]
print (lst)
```

A. 1
B. 4
C. 2
D. 3

Correct Answer: C
Explanation

Explanation/Reference:
`lst[-3:-2]` means from the 3rd element counting from last (inclusive) to the 2nd element counting from last(exclusive)
Only one element "2" is included in the new List object (i.e. "[2]") and the new List object is assigned back to "lst"

Since "lst" only have one element "2", lst[-1] is therefore the last element is the only element i.e. 2. Since [-1] is not a Slice notation, the value "2" (instead of a List object) is returned and assigned back to lst.

Code for testing:

```
lst = [1,2,3,4]
lst = lst[-3:-2]
print(lst)  # [2]
lst = lst[-1]
print(lst)  # 2  <-- final result
```

QUESTION 17
What is the expected output of the following snippet?

```
s= 'abc'
for i in len(s):
    s[i] = s[i].upper ( )
print(s)
```

A. abc
B. The code will cause a runtime exception
C. ABC
D. 123

Correct Answer: B
Explanation

Explanation/Reference:
Since String is immutable, you cannot change a character in a string "s" through `s[i] = .....`

QUESTION 18
How many elements will the list2 list contain after execution of the following snippet?

```
list1 = [False for i in range (1, 10) ]
list2 = list1 [-1:1:-1]
```

A. zero
B. five
C. seven
D. three

Correct Answer: C
Explanation

Explanation/Reference:
`[False for i in range (1, 10)]` will generates 9 False elements in a new List.
`list1[-1:1:-1]` will slice the element in reverse order from last element (inclusive) to the 2nd element (exclusive) i.e. all except the first two elements in index 0 and 1. Therefore the total number of elements i.e. False will be 9 - 2 = 7.

Code for testing:

```
list1 = [False for i in range (1, 10)]
list2 = list1[-1:1:-1]
print(len(list2))   # 7

# extra codes for easier understanding
print()
list11 = [i for i in range (1, 10)]
print(list11)        # [1, 2, 3, 4, 5, 6, 7, 8, 9]
list12 = list11[-1:1:-1]
print(len(list12))   # 7
print(list12)        # [9, 8, 7, 6, 5, 4, 3]
```

QUESTION 19
What would you used instead of XXX if you want to check wether a certain 'key' exists in a dictionary called dict? (Choose two.)

```
if XXX:
    print   Key exists
```

A. "key" in dict
B. dict["key"] != None
C. dict.exists("key")
D. "key "in dict.keys()

Correct Answer: AD
Explanation

Explanation/Reference:
Code for testing:

```
dict1 = {"key" : 1, "kkk" :2}
print("key" in dict1)         # True.  Using dict1 directly will match the keys
only.
print("aaa" in dict1)         # False
print("key" in dict1.keys())  # True
print("aaa" in dict1.keys())  # False
print()
print(dict1["key"] != None)    # ok if key exists BUT will cause error if key
not exists
# print(dict1["aaa"] != None)  # Error is key not exists
```

```
# print(dict.exists("key"))    # method not exist
```

QUESTION 20
You need data which can act as a simple telephone directory. You can obtain it with the following clauses (Choose two.) (assume that no other items have been created before)

A. `dir={"Mom",5551234567:"Dad",557654321}`
B. `dir={"Mom","5551234567":"Dad","557654321"}`
C. `dir={"Mom":5551234567,"Dad":557654321}`
D. `dir={"Mom":"5551234567","Dad":"557654321"}`

Correct Answer: CD
Explanation

Explanation/Reference:
Colon ":" must be used for seperating key and value for the same entry.
Comma "," must be used for seperating different entries.

QUESTION 21
Can a module run like regular code?

A. yes, and it can differentiate its behavior between the regular launch and import
B. it depends on the Python version
C. yes, but in cannot differentiate its behavior between the regular launch and import
D. no, it is not possible; a module can be imported, not run

Correct Answer: A
Explanation

Explanation/Reference:
You can use the value returns from `__name__` in a `if` statement for the differentiation.

QUESTION 22
Select the valid fun () invocations:
(Choose two.)

```
def fun (a, b=0):
    return a*b
```

A. fun(b=1)
B. fun(a=0)
C. fun(b=1, 0)
D. fun(1)

Correct Answer: BD
Explanation

Explanation/Reference:
Based on the followings:
- argument for the parameter "a" must be supplied (A is wrong because of this)
- all Positional Argument must be supplied before any Keyword Argument (C is wrong because of this)

Remarks:
All the following are valid:

```
def fun(a,b=0):
~~~~return a*b
```

```
fun(0,1)
fun(0)
fun(0, b=1)
fun(a=0)
fun(a=0,b=1)
fun(b=1,a=0)
```

QUESTION 23
A file name like this one below says that :(Choose three.)
`services.cpython-36.pyc`

A. the interpreter used to generate the file is version 3.6
B. it has been produced by CPython
C. it is the 36 version of the file
D. the file comes from the services.py source file

Correct Answer: ABD
Explanation

Explanation/Reference:

QUESTION 24
What is the expected behavior of the following snippet?

```
def a(I, l):
    return l[I]

print(a(0,[1]))
```

It will:

A. cause a runtime exception
B. print 1
C. print 0, [1]
D. print [1]

Correct Answer: B
Explanation

Explanation/Reference:
0 and List object [1] are passed as two arugments to the function `a( )`
The function return the index 0 (obtained from first argument) of the List object (obtained from second argument). i.e. 1.

Code for test:

```
def a(I, l):
~~~~return l[I]
print(a(0,[1]))  # 1
```

Remarks :
If the position of the two parameters are swapped, error will occur since you cannot use an index number on `int` value.
`def a(l, I):    # <-- position for the two parameters are swapped`

```
    return l[I]
```

QUESTION 25
What can you do if you don't like a long package path like this one?

```
import alpha .beta . gamma .delta .epsilon .zeta
```

A. you can make an alias for the name using the `alias` keyword
B. nothing, you need to come to terms with it
C. you can shorten it to `alpha.zeta` and Python will find the proper connection
D. you can make an alias for the name using the `as` keyword

Correct Answer: D
Explanation

Explanation/Reference:
For example, `import alpha.beta.gamma.delta.spilon.zeta as zeta`

Remarks :
For B, "come to terms with" means you can only accept it although you do not like it

QUESTION 26
What is the expected output of the following code?

```
str = 'abcdef'
def fun (s) :
    del s [2]
    return s

print (fun (str) )
```

A. abcef
B. The program will cause a runtime exception/error
C. acdef
D. abdef

Correct Answer: B
Explanation

Explanation/Reference:
String is immutable and therefore you cannot remove a character in an existing String object with `del s [2]`.

QUESTION 27
What is the expected output of the following code?

```
def f (n) :
    if n == 1:
        return '1'
    return str (n) + f (n-1)

print (f (2) )
```

A. 21
B. 2
C. 3
D. 12

Correct Answer: A
Explanation

Explanation/Reference:
In the recursive calling:
`f(2)` will return "2" + `f(1)`
`f(1)` will return "1"

Hence the result is the string concatenation "2" + "1" i.e. "21".

Code for testing:

```
def f(n):
~~~~if n == 1:
~~~~~~~~return '1'
~~~~return str(n) + f(n-1)

print(f(2))
```

QUESTION 28
What is the expected behavior of the following snippet?

```
def x( ) :          # line 01
    return 2        # line 02

x= 1 + x ( )        # line 03
print (x)           # line 04
```

A. cause a runtime exception on line 02
B. cause a runtime exception on line 01
C. cause a runtime exception on line 03
D. print 3

Correct Answer: D
Explanation

Explanation/Reference:
The program runs normally since the function `x( )` is called before the assignment that reassigns an `int` value to the variable `x`.
Since x() returns 2 and then add with 1, the result 3 is assigned to "x" and printed.

Code for testing:

```
def x():
~~~~return 2

x = 1 + x()
print(x)    # 3
```

QUESTION 29
What is the expected behavior of the following code?

```
def f (n):
    for i in range (1, n+1) :
        yield I

print (f(2) )
```

It will:

A. print 4321

B. print <generator object f at (some hex digits)>
C. cause a runtime exception
D. print 1234

Correct Answer: B
Explanation

Explanation/Reference:
Since the function contains yield, it will returns a generator object.
Values from yield will only returns if you using call `next()` using the generator object as arugment.

Code for testing:

```
def f(n):
~~~~for i in range(1, n+1):
~~~~~~~~yield i

print(f(2))   # print generator object

# valid way to get the int values from yield
g = f(2)
print(next(g))  # 1
print(next(g))  # 2
print(next(g, "no more"))  # no more
```

QUESTION 30
If you need a function that does nothing, what would you use instead of XXX? (Choose two.)

```
def idler ( ):
    XXX
```

A. pass
B. return
C. exit
D. None

Correct Answer: AB
Explanation

Explanation/Reference:
Acutally all of them is valid syntax. The progam can run without causing error.
However, A and B are usually used. They are well understood by all programmers for achieving the purpose of doing nothing.

For C, `exit()` represents a function which is used for exiting the entire Python program.
However, in the choice C, the "`exit`" is used without bracket and this represents the function object.
Entering this alone in the statement will not affect program running and do nothing. If it is the last statement, the function will then exit.

For D, None is a value of type Nonetype. Puting this alone will not cause error and will not affect program running. If it is the last statement, the function will then exit.

Hence, C and D is just like entering "abc" in the following second line.

```
abc = 3
abc          # <------
print(abc)   #  will print 3.
```

It will not affect the program and will not cause error.

However, these statements "exit", "None", "abc" are not recommended in the program code since they may cause confusion. Other programmers reading your program codes may think that something is missing in the statement.

QUESTION 31
Is it possible to safely check if a class/object has a certain attribute?

A. yes, by using the hasattr attribute
B. yes, by using the hasattr( ) method
C. yes, by using the hasattr( ) function
D. no, it is not possible

Correct Answer: C
Explanation

Explanation/Reference:
It is safe since `False` will be returned if the attribute does not exist i.e. will not cause error.
Note that "hasattr()" is a built-in function.
It is not an attribute and method i.e. it is called without any qualifier.

QUESTION 32
The first parameter of each method:

A. holds a reference to the currently processed object
B. is always set to None
C. is set to a unique random value
D. is set by the first argument's value

Correct Answer: A
Explanation

Explanation/Reference:
Since the syllabus only includes Instance method, therefore the first parameter here should means the one we usually use the name "`self`".

QUESTION 33
The simplest possible class definition in Python can be expressed as:

A. class X:
B. class X: pass
C. class X: return
D. class X: { }

Correct Answer: B
Explanation

Explanation/Reference:
There must be a suite under the header. Unlike a function, return cannot be used directly under a class definition.

QUESTION 34
If you want to access an exception object's components and store them in an object called e, you have to use the following form of exception statement:

A. `except Exception(e):`
B. `except e = Exception:`
C. `except Exception as e:`
D. such an action is not possible in Python

Correct Answer: C
Explanation

Explanation/Reference:

QUESTION 35
A variable stored separately in every object is called:

A. there are no such variables, all variables are shared among objects
B. a class variable
C. an object variable
D. an instance variable

Correct Answer: D
Explanation

Explanation/Reference:
The term "instance variable" is the official naming for this kind of variable. Therefore C is wrong.

QUESTION 36
There is a stream named s open for writing. What option will you select to write a line to the stream?

A. `s.write("Hello")`
B. `write("Hello")`
C. `s.writeln("Hello")`
D. `s.writeline("Hello")`

Correct Answer: A
Explanation

Explanation/Reference:
The only valid choice is A.
B is wrong since there is no qualifier to call the write() method
C and D is wrong is there is no such method. (The only method with similar name is `writelines()` which is ending with a "s")

QUESTION 37
You are going to read just one character from a stream called s. Which statement would you use?

A. ch = read (s, 1)
B. ch = s.input (1)
C. ch = input (s, 1)
D. ch = s.read (1)

Correct Answer: D
Explanation

Explanation/Reference:
A is wrong since read() accepts only one argument and must be called with a qualifier.
B is wrong since input() is a build-in function for accepting user input from stdin without any qualifier.
C is wrong since input() accepts only one argument and it accepts user input from stdin.

QUESTION 38
What can you deduce from the following statement? (Choose two.)

```
str= open ('file.txt', 'rt')
```

A. `str` is a string read in from the file named file.txt
B. a newline character translation will be performed during the reads
C. if file.txt does not exist, it will be created
D. the opened file cannot be written with the use of the `str` variable

Correct Answer: BD
Explanation

Explanation/Reference:
A is wrong since `str` is a File stream object.
C is wrong since mode "r" will not create file.

Remarks (more about the choice B):
In Python3, when a file is opened for Read in text mode, the newline character in the file may be translated into a standard form.
For example, if a text file is created by Windows, the newline character is "\r\n" in the file will be translated to "\n".
This allows Python can get the same content no matter which OS is used for creating the text file.

QUESTION 39
The following class hierarchy is given. What is the expected out of the code?

```
class A:
    def a (self) :
        print (“A”, end= ‘ ’)
    def b (self) :
        self.a ( )

class B (A):
    def a (self) :
        print (“B”, end= ‘ ’)
    def do (self):
        self.b ( )

class C (A):
    def a (self):
        print (“C”, end= ‘ ’)
    def do (self):
        self.b ( )

B ( ) . do ( )
C ( ) . do ( )
```

A. BB
B. CC
C. AA
D. BC

Correct Answer: D
Explanation

Explanation/Reference:
`B().do()`
B object is created and it is used as qualifier to call `do()`:
- The `do()` of class B is run and it calls `self.a()`
- Since "self" is the B object, the `a()` of class B is run and it prints **B**.

`C().do()`
C object is created and it is used as qualifier to call `do()`:
- The `do()` of class C is run and it calls `self.a()`
- Since "self" is the C object, the `a()` of class C is run and it prints **C**

Code for testing:

```
class A:
~~~~def a(self):
~~~~~~~~print ("A", end='')
~~~~def b(self) :
~~~~~~~~self.a( )

class B(A):
~~~~def a(self):
~~~~~~~~print ("B", end='')
~~~~def do(self) :
~~~~~~~~self.a()

class C(A):
~~~~def a(self):
~~~~~~~~print ("C", end='')
~~~~def do(self):
~~~~~~~~self.a( )

B().do()   # prints B  (without newline character appended)
C().do()   # prints C  (without newline character appended)
```

QUESTION 40
Python's built in function named open() tries to open a file and returns:

A. an integer value identifying an opened file
B. an error code (0 means success)
C. a stream object
D. always None

Correct Answer: C
Explanation

Explanation/Reference:

QUESTION 41
Which of the following words can be used as a variable name? (Choose two.)

A. for
B. True
C. true
D. For

Correct Answer: CD
Explanation

Explanation/Reference:
A is wrong since "for" is a keyword
B is wrong since "True" is a Boolean Literal.

QUESTION 42
Python strings can be `glued` together using the operator:

A. .
B. &
C. _
D. +

Correct Answer: D
Explanation

Explanation/Reference:
Only "+" is allows for gluing two strings.
"*" can be used for string. However, one operand must be `str` and the other operand must be `int` value.

QUESTION 43
A keyword (Choose two.)

A. can be used as an identifier
B. is defined by Python's lexis
C. is also known as a reserved word
D. cannot be used in the user's code

Correct Answer: BC
Explanation

Explanation/Reference:

QUESTION 44
How many stars (*) does the snippet print?

```
s = '*****'
s = s - s [2]
print (s)
```

A. the code is erroneous
B. five
C. four
D. two

Correct Answer: A
Explanation

Explanation/Reference:
The operator "-" cannot be used for String operation

QUESTION 45
Which line can be used instead of the comment to cause the snippet to produce the following expected output? (Choose two.)
Expected output:
1 2 3

Code:

```
c, b, a = 1, 3, 2
# put line here
print (a, b, c)
```

A. c, b, a = b, a, c
B. c, b, a = a, c, b
C. a, b, c = c, a, b
D. a, b, c = a, b, c

Correct Answer: AC
Explanation

Explanation/Reference:
After the first statement, a is 2, b is 3 and c is 1
In order for a b c to print 1 2 3:
- a needs to be 1 i.e. assigned with value in c i.e. a = c
- b needs to be 2 i.e. assigned with value in a i.e. b = a
- c needs to be 3 i.e. assigned with value in b i.e. c = b

In order to prevent one assignment of the above from affecting the value being assigned in other assignments, they should be assigned at the same time in the same statement. Within a single statement, the order of the assignment in the statement is not important. Hence A and C are both correct.

Code for testing:

```
c, b, a = 1, 3, 2
print(a, b, c)  # 2 3 1

c, b, a = 1, 3, 2
c, b, a = b, a, c   # assigning 3 2 1
print(a, b, c)  # 1 2 3   (correct answer)

c, b, a = 1, 3, 2
c, b, a = a, c, b   # assigning 2 1 3
print(a, b, c)  # 3 1 2

c, b, a = 1, 3, 2
a, b, c = c, a, b   # assigning 1 2 3
print(a, b, c)  # 1 2 3   (correct answer)

c, b, a = 1, 3, 2
a, b, c = a, b, c   # assigning 2 3 1
print(a, b, c)  # 2 3 1
```

QUESTION 46
Assuming that the V variable holds an integer value to 2, which of the following operators should be used instead of OPER to make the expression equal to 1?
V OPER 1

A. <<<
B. >>>
C. >>
D. <<

Correct Answer: C
Explanation

Explanation/Reference:
In the following , leading zeros are added to show the operation more clearly.
The value 2 in binary format is 0010
The value 1 in binary format is 0001

Hence, the bit "1" has to be shifted to the right by 1 bit.
Therefore the operator that should be used in the location "OPER" is ">>" (arrow pointing to the right).

Code for testing:

```
V = 2
print(V >> 1)   # 1
```

QUESTION 47
How many stars (*) does the following snippet print?

```
i = 3
while i > 0 :
        i -= 1
        print ("*")
else:
        print ("*")
```

A. the code is erroneous
B. five
C. three
D. four

Correct Answer: D
Explanation

Explanation/Reference:
Since `i` is set to 3, the condition `i > 0` in `while` returns True.
i) `i-=1` will subtract 1 from `i` and `i` becomes 2.
ii) a "*" is printed.

Since `i` is now 2, the condition `i > 0` in `while` still returns True.
i)`i-=1` will subtract 1 from `i` and `i` becomes 1.
ii) a "*" is printed.

Since `i` is now 1, the condition `i > 0` in `while` still returns True.
i) `i-=1` will subtract 1 from `i` and `i` becomes 0.
ii) a "*" is printed.

Since `i` is now 0, the condition `i > 0` in `while` returns False.
This triggers the running of `else` clause which prints a "*".

Program ends. A total of four "*" is printed.

Code for testing:

```
i = 3
while i > 0 :
~~~~i -= 1
~~~~print("*")
else:
~~~~print("*")
```

QUESTION 48
UNICODE is:

A. the name of an operating system
B. a standard for encoding and handling texts
C. the name of a programming language
D. the name of a text processor

Correct Answer: B
Explanation

Explanation/Reference:
UNICODE a standard for encoding and handling texts. UTF-8 is one of the encoding format of UNICODE

standard and is used by default in Python.

QUESTION 49
What is the expected output of the following snippet?

```
s = '* - *'
s = 2* s + s* 2
print (s)
```

A. `*-**-**-**-*`
B. `*-**-**-**-**-**-**-**-*`
C. `*-*`
D. `*-**-*`

Correct Answer: A
Explanation

Explanation/Reference:
Like arithematic operations, "*" is evaluated before "+".

`'*-*'` concatenating to itself 2 times is `'*-**-*'`
Two of the above concatenate together by "+" will be `'*-**-**-**-*'`

Code for testing:

```
s = '*-*'
s = 2 * s + s * 2
print(s)
```

QUESTION 50
Which of the listed actions can be applied to the following tuple? (Choose two.)

```
tup = ()
```

A. tup[:]
B. tup.append (0)
C. tup[0]
D. del tup

Correct Answer: AD
Explanation

Explanation/Reference:
B is wrong since tuple is immutable
C is wrong since there is no element in the Tuple and this causes index out of range.

A is correct since you are getting elements from Tuple (although it is empty)
D is correct since you are deleting the variable "tup" instead of deleting elements within a Tuple.

QUESTION 51
Executing the following snippet -

```
dct = { 'pi' : 3.14}
dct ['pi'] = 3.1415
```

will cause the dict:

A. to hold two keys named 'pi' linked to 3.14 and 3.1415 respectively

B. o hold two key named 'pi' linked to 3.14 and 3.1415
C. to hold one key named 'pi' linked to 3.1415
D. to hold two keys named 'pi' linked to 3.1415

Correct Answer: C
Explanation

Explanation/Reference:
The assignment in the second statement replace the value 3.14 with 3.1415 for the key "pi".

QUESTION 52
How many elements will the list1 list contain after execution of the following snippet?

```
List1 = "don't think twice, do it!" .split (',')
```

A. two
B. zero
C. one
D. three

Correct Answer: A
Explanation

Explanation/Reference:
Since the separator "," only occurs once in the String, only two Substring will be splitted for forming the elements in the new List object. Therefore the new List object contains only two elements

Code for testing:

```
List1 = "dont't think twice, do it!".split(',')
print(len(List1))   # 2
print(List1)        # ["dont't think twice", ' do it!']
```

QUESTION 53
Which of the equations are True? (Choose two.)

A. chr(ord(x)) == x
B. ord(ord(x)) == x
C. chr(chr(x)) == x
D. ord(chr(x)) == x

Correct Answer: AD
Explanation

Explanation/Reference:
chr() and ord() can be used to convert between code point and character.

If x is a `int` vlaue, then D applies. The `int` value is converted into a character by `chr()` and then convert back into `int` value by `ord()`
If x is a character, the A applies. The character is converted into a int value by `ord()` and then convert back into a character.by `chr()`

QUESTION 54
If you want to transform a string into a list of words, what invocation would you use? (Choose two.)
Expected output:

```
The, Catcher, in, the Rye,
```

Code:

```
S = "The Catcher in the Rye"
1 = # put a proper invocation here
For w in 1:
    Print (w, end=',') # outputs: The, Catcher, in, the Rye,
```

A. `s.split()`
B. `split(s, ",")`
C. `s.split(" ")`
D. `split(s)`

Correct Answer: AC
Explanation

Explanation/Reference:
Among the choices, only two (A and C) are valid statement since split() is an instance method of String and therefore must be called by using the concerned string as qualifier i.e. the syntax "`s.split(....)`"

Remarks:
There is something wrong in the question / choices since the using of default value `None` or " " as sep will splitted the String to a List object as follows:
`['The', 'Catcher', 'in', 'the', 'Rye']`

The output from "`print()`" will therefore be as follows.
`The, Catcher, in, the, Rye,`

In order to combine "the" and Rye", you should also limit the number of splitting to use only 3 separators only e.g.
`s.split(`**`maxsplit=3`**`)` or `s.split("",`**`maxsplit=3`**`)`

Code for testing:

```
s = "The Catcher in the Rye"
l = s.split(" ")
print(l)
for w in l:
~~~~print(w, end=', ')              # The, Catcher, in, the, Rye,
l = s.split(" ", maxsplit=3)
for w in l:
~~~~print(w, end=', ')              # The, Catcher, in, the Rye,
```

QUESTION 55
Assuming that `lst` is a four-element list, is there any difference between these two statements?

```
del 1st # the first line
del 1st [:] # the second line
```

A. yes, there is, the first line empties the list, the second line deletes the list as a whole
B. yes, there is, the first line deletes the list as a whole, the second line just empties the list
C. no, there is no difference
D. yes, there is, the first line deletes the list as a whole, the second line removes all the elements except the first one

Correct Answer: B
Explanation

Explanation/Reference:
Code for testing.

```
lst1 = [1, 2, 3, 4]
lst2 = [1, 2, 3, 4]
del lst1
del lst2[:]
print("lst1" in vars())  # False. lst1 not exist any more
print("lst2" in vars())  # True. lst2 still exists
print(type(lst2))        # <class 'list'>
print(len(lst2))         # 0  i.e. An empty List
```

QUESTION 56
What should you put instead of XXX to print out the module name?

```
If ____name____ ! = "XXX":
    print (__name__)
```

A. `main`
B. `_main_`
C. `__main__`
D. `___main___`

Correct Answer: C
Explanation

Explanation/Reference:
two underscores before and after "main".

QUESTION 57
Files with the suffix .pyc contain:

A. Python 4 source code
B. backups
C. temporary data
D. semi-compiled Python code

Correct Answer: D
Explanation

Explanation/Reference:

QUESTION 58
Package source directories/folders can be:

A. converted into the so-called pypck format
B. packed as a ZIP file and distributed as one file
C. rebuilt to a flat form and distributed as one directory/folder
D. removed as Python compiles them into an internal portable format

Correct Answer: B
Explanation

Explanation/Reference:

QUESTION 59
What can you deduce from the line below? (Choose two.)

```
x = a.b.c.f ()
```

A. `import a.b.c` should be placed before that line
B. `f()` is located in subpackage c of subpackage b of package a
C. the line is incorrect
D. the function being invoked is called `a.b.c.f()`

Correct Answer: AB
Explanation

Explanation/Reference:
D is wrong since dot "." cannot be used to form part of a function name.

QUESTION 60
A two-parameter lambda function raising its first parameter to the power of the second parameter should be declared as:

A. lambda (x, y) = x ** y
B. lambda (x, y): x ** y
C. def lambda (x, y): return x ** y
D. lambda x, y: x ** y

Correct Answer: D
Explanation

Explanation/Reference:
A, B and C are wrong is braces not allowed to enclose parameters
Moreover:
- A is also wrong due to the use of "=" instead of ":".
- C is also wrong since "def" and "return" should not be entered.

QUESTION 61
What is the expected output of the following code?

```
def f (n):
if n == 1:
return 1
return n + f (n-1)
print (f(2))
```

A. 21
B. 12
C. 3
D. none

Correct Answer: C
Explanation

Explanation/Reference:
The identation in the picture in this question is wrong (properly due to typing mistake).

Actually the syntax of the program code should be similar to that in Q27 except that `'1'` and `str()` and `'1'` are used in the two returnsin Q27. In this question, all values returned are in numeric and therefore "+" will perform addtiion.

Therefore:

In Q27, `"2" + "1" ---> "21"`
In this question, `2 + 1 ---> 3`

Code for testing:

```
def f(n):
~~~~if n == 1:
~~~~~~~~return 1
~~~~return n + f(n-1)

print(f(2))
```

QUESTION 62
A method for passing the arguments used by the following snippet is called:

```
def fun (a, b):
      return a + b

res = fun (1, 2)
```

A. sequential
B. named
C. positional
D. keyword

Correct Answer: C
Explanation

Explanation/Reference:

QUESTION 63
What is the expected behavior of the following code?

```
def f(n):
for i in range (1, n+1):
yield i

for i in f (2):
    print (i, end= ' ')
```

It will:

A. print 2 1
B. print 1 2
C. cause a runtime exception
D. print <generator object f at (some hex digits)>

Correct Answer: B
Explanation

Explanation/Reference:
The identation in this picture is properly wrong due to typing mistake.

For calling `f(2)`, the function will be used to iterate number from 1 (inclusive) to 2+1 i.e. 3 (exclusive) Hence iteration will be performed with two numbers: 1 and 2 for returning through by each `yield`

In the main program code, the `for i in f(2)` will obtain each number from `yield` and print it with a space appended at the end. Hence the result will be:

1<space>2<space>

Code for testing:

```
def f(n):
~~~~for i in range(1, n+1):
~~~~~~~~yield i

for i in f(2):
~~~~print(i, end=' ')
```

QUESTION 64
What is the expected output of the following code?

```
1st = [x for x in range (5)]
1st = list (filter (lambda x: x % 2 = = 0, 1st))
print (len(1st))
```

A. 2
B. The code will cause a runtime exception
C. 1
D. 3

Correct Answer: D
Explanation

Explanation/Reference:
A List object "`lst`" is formed using List Comprehension literal having elements 0, 1, 2, 3, 4

`filter()` uses a Lambda function for finding even number for filtering from `lst`
A new list is formed from the `filter` object having elements 0, 2, 4 and assigned back to `lst`
Hence the number of elements in the `lst` is now 3.

Code for testing:

```
lst = [x for x in range (5)]
lst = list (filter (lambda x: x % 2 == 0, lst))
print (len(lst))
```

QUESTION 65
What is the expected behavior of the following code?

```
def unclear (x):
    if x % 2 = = 1:
       return 0

print )unclear (1) + unclear (2))
```

It will:

A. print 0
B. cause a runtime exception
C. prints 3
D. print an empty line

Correct Answer: B
Explanation

Explanation/Reference:
This picture has typing mistake about an opening bracket and the comparison equal.

When `unclear(1)` is called, since `1 % 2` has a reminder of 1, the condition of `if` returns `True`. Therefore 0 is returned.

When `unclear(2)` is called, since `2 % 2` has no reminder (i.e. 0), the condition of `if` returns `False`. Since this is no more statement for running in the function, the function ends with a default return value `None`..

In the main program code, it try to print the result of `unclear(1)` + `unclear(2)`. However, you cannot use the operator "+" when one operand is 0 and the other operand is `None`. Hence error will occur.

Code for testing:

```
def unclear(x):
~~~~if x * 2 == 1:
~~~~~~~~return 0

print(unclear(1) + unclear(2))
```

QUESTION 66
If any of a class's components has a name that starts with two underscores (___), then:

A. the class component's name will be mangled
B. the class component has to be an instance variable
C. the class component has to be a class variable
D. the class component has to be a method

Correct Answer: A
Explanation

Explanation/Reference:

QUESTION 67
If you need to serve two different exceptions called Ex1 and Ex2 in one except branch, you can write:

A. except Ex1 Ex2:
B. except (Ex1, Ex2):
C. except Ex1, Ex2:
D. except Ex1+ Ex2:

Correct Answer: B
Explanation

Explanation/Reference:

QUESTION 68
A function called `issubclass(c1, c2)` is able to check if:

A. c1 and c2 are both subclasses of the same superclass
B. c2 is a subclass of c1
C. c1 is a subclass of c2
D. c1 and c2 are not subclasses of the same superclass

Correct Answer: C
Explanation

Explanation/Reference:

QUESTION 69
Which of the following lambda function definitions are correct? (Select two answers)

A. `lambda x : None`
B. `lambda : 3.1415`
C. `lambda x: def fun(x): return x`
D. `lambda lambda: lambda * lambda`

Correct Answer: AB
Explanation

Explanation/Reference:
C is wong since only one ":" is allowed. Moreover, on its right, it must be an expression instaed of a complete statement "return x"
D is wrong since "`lambda`" is a keyword / reserved word and cannot be used as parameter variable name

QUESTION 70
What is true about the following snippet? (Select two answers)

```
class E(Exception):
    def __init__(self, message):
        self.message = message
    def __str__(self):
        return "it's nice to see you"


try:
    print("I feel fine")
    raise Exception("what a pity")
except E as e:
    print(e)
else:
    print("the show must go on")
```

A. the code will raise an unhandled exception
B. the string `I feel fine` will be seen
C. the string `it's nice to see you` will be seen

D. the string `what a pity` will be seen

Correct Answer: AB
Explanation

Explanation/Reference:
The 1st statement in the try clause is run and the string `I feel fine` is printed
The 2nd statement in the try clause raise an Exception.

However, the `except` clause only catch the custom Exception "E" which is not a SuperClass of Exception. Therefore the exception cannot be handled and the program aborts.

Remarks (about modified version of this qustion) :
If the "`raise`" statement is changed to trigger the custom exception. Then B and C will be the answer. Since the `__str__` in the custom exception overrides the parent's implementation. The returned string `"it's nice to see you"` is printed by `print(e)` in the `except` suite.

```
class E(Exception):
~~~~def __init__(self, message):
~~~~~~~~self.test = message
~~~~def __str__(self):
~~~~~~~~return "it's nice to see you"

try:
~~~~print("I feel fine")
~~~~raise E("what a pity")   # Already changed from Exception to E
except E as e:
~~~~print(e)
else:
~~~~print("the show must go on")
```

Additional Reference:
The `__init__` in custom exception "E" is not required. It is needed in Python 2.5 or before. Moreover, later Python version uses `args` instead of `message`.

Note that calling `super().__init__()` is not required in "__init__" unless you want to change the arugments passed.
This is special since unlike other classes, BaseException has a "`__new__`" and it uses it to assign arugment to "`args`". Since this "`__new__`" will be inherited by all subclasses by default, arguments can be set to "`args`" when creating a Exception object without call "`super().__init__()`".

```
class EE(Exception):
~~~~def __init__(self, message):
~~~~~~~~pass              # super().__init__() is not called

try:
~~~~print("I feel fine")
~~~~raise EE("what a pity")
except EE as e:
~~~~print(e)              # can still print "what a pity"
else:
~~~~print("the show must go on")
```

QUESTION 71
Which of the following expression evaluate to True? (Select two answers)

A. `'in not' in 'not'`
B. `'in' in 'Thames'`
C. `'t'.upper() in 'Thames'`
D. `'in' in 'in'`

Correct Answer: CD

Explanation

Explanation/Reference:
C is correct since `'t'.upper()` i.e. `'T'` can be found within `'Thames'`
D is correct since `'in'` can be found within `'in'`

Code for testing:
```
print('in not' in 'not')        # False
print('in' in 'Thames')         # False
print('t'.upper() in 'Thames')  # True
print('in' in 'in')             # True
```

QUESTION 72
Which of the following snippets will execute without raising any unhandled exceptions? (Select 2 answers)

A.
```
try:
    print(int("0"))
except NameError:
    print("0")
else:
    print(int(""))
```

B.
```
try:
    print(0/0)
except:
    print(0/1)
else:
    print(0/2)
```

C.
```
import math

try:
    print(math.sqrt(-1))
except:
    print(math.sqrt(0))
else:
    print(math.sqrt(1))
```

D.
```
try:
    print(float("1e1"))
except (NameError, SystemError):
    print(float("1a1"))
else:
    print(float("1c1"))
```

Correct Answer: BC
Explanation

Explanation/Reference:
For A, the statement in `try` clause does not raise any exception. However, the statement in the `else` clause raises an exception. Since it it not within a `try` clause, it will not be handled.

For B, the statement in `try` clause raise an exception but is handled by the default "BaseException". The statement in the `except` clause does not raise any exception. Therefore, there is no unhandled exception.

For C, the statement in `try` clause raise an exception but is handled by the default "BaseException". The statement in the `except` clause does not raise any exception. Therefore, there is no unhandled exception.

For D, the statement in `try` clause does not raise any exception. However, the statement in the `else` clause raises an exception. Since it it not within a `try` clause, it will not be handled.

Code for testing
(Program codes for A and D are enclosed by another `try` statement and prints traceback about the unhandled exception within).

```
import traceback

print("Choice A:")
try:
~~~~try:
~~~~~~~~print(int("0"))
~~~~except NameError:
~~~~~~~~("0")
```

```
~~~~else:
~~~~~~~~print(int(""))   # Unhandled Error!
except:
~~~~print(traceback.format_exc())
print()

print("Choice B:")
try:
~~~~print(0/0)
except:
~~~~print(0/1)
else:
~~~~print(0/2)
print()

print("Choice C:")
import math
try:
~~~~print(math.sqrt(-l))
except:
~~~~print(math.sqrt(0))
else:
~~~~print(math.sqrt(1))
print()

print("Choice D:")
try:
~~~~try:
~~~~~~~~print(float("1e1"))
~~~~except (NameError, SystemError):
~~~~~~~~print(float("1a1"))
~~~~else:
~~~~~~~~print(float("1c1"))   # Unhandled Error!
except:
~~~~print(traceback.format_exc())
```

QUESTION 73
Which of the following invocations are valid? (Select two answers)

A. `find("python","r")`
B. `sorted("python")`
C. `"python".sort()`
D. `"python".index("th")`

Correct Answer: BD
Explanation

Explanation/Reference:
A is wrong since find() is a instance method and therefore must be called with a String object e.g. "`s.find ( ...)`"
C is wrong since String does not have the `sort()` method.

B is valid since `sorted()` is a built-in function and do not use String object to call it.
D is valid since `index()` is a instance method and is called with a String object.

QUESTION 74
What is the expected behavior of the following code?

```
class Class:
    _Var = 1
    __Var = 2
    def __init__(self):
        self._prop = 3
        self.__prop = 4


o = Class()
print(o._Class__Var + o._Class__prop)
```

A. it outputs 6
B. it outputs 1
C. it outputs 3
D. it raises an exception

Correct Answer: A
Explanation

Explanation/Reference:
For easy understanding, you can convert the name involves in name mangling fwithin the class first:

```
class Class:
    _Var = 1
    _Class__Var = 2
    def __init__(self):
        self._prop = 3
        self._Class__prop = 4
```

Since _Class__Var is a class variable and can be accessed by "Class." or "o."
Therefore the answer is 2 + 4 = 6.

```
class Class:
~~~~_Var = 1
~~~~__Var = 2
~~~~def __init__(self):
~~~~~~~~self._prop = 3
~~~~~~~~self.__prop = 4

o = Class()
print(o._Class__Var + o._Class__prop)
```

QUESTION 75
What is the expected behavior of the following code?

```
string = str(1/3)
dummy = ''
for character in string:
    dummy = dummy + character
print(dummy[-1])
```

A. it outputs 'None'
B. it outputs 3
C. it raises an exception
D. it outputs 0

Correct Answer: B
Explanation

Explanation/Reference:
The above converts the float value 1/3 i.e. 0.333..... to a string and then use for statement to get each of the character.
Each of the character is concatenated to the string "dummy" which is initially an empty String.
Finally, the last characters in the string "dummy" is printed.

Since the dummy will contain exactly the same content as str(1/3), the last character is therefore 3.

Code for testing:

```
string = str(1/3)
dummy = ''
for character in string:
~~~~dummy = dummy + character
print(dummy)        # 0.3333333333333333
print(dummy[-1])    # 3
```

QUESTION 76
What is the expected behavior of the following code?

```
x = 8 ** (1/3)
y = 2. if x < 2.3 else 3.
print(y)
```

A. it outputs 2.0
B. it outputs 2.5
C. the code is erroneus and it will not execute
D. it outputs 3.0

Correct Answer: A
Explanation

Explanation/Reference:
The cube root of 8 is 2. (Since 2 to the power of 3 i.e. 2 * 2 * 2 = 8). Hence, x is assigned with 2.

Since `x < 2.3` is True, the if ... else ... expression returns the first value i.e. `2.` (a float value since there is a dot after "2") for assigning to `y`.

Hence, printing "y" will show "2.0".

You can copy and paste the code in the question directly for testing.

QUESTION 77
Assuming that the code below has been executed successfully, which of the following expressions will always evaluate to True? (Select two answers)

```
import random
v1 = random.random()
v2 = random.random()
```

A. len(random.sample([1,2,3],1)) > 2
B. v1 == v2
C. random.choice([1,2,3]) > 0
D. v1 < 1

Correct Answer: CD
Explanation

Explanation/Reference:
A is wrong since `random.sample([1,2,3],1)` returns 1 element chosen from 1,2,3. Therefore the `len()` of it must be 1.
B is wrong since v1 and v2 are two different random numbers.

C is correct since `random.choice([1,2,3])` must be either 1, 2 or 3.
D is corrent since v1 which is a value from `random()` must be >= 0 and < 1

Code for testing:

```
import random
v1 = random.random()
v2 = random.random()

print(len(random.sample([1,2,3],1)) > 2)    # False
print(v1 == v2)                             # False
print(random.choice([1,2,3]) > 0)           # True
print(v1 < 1)                               # True
```

QUESTION 78
What is the expected output of the following code if the file named "zero_length_existing_file" is a zero-length file located inside the working directory?

```
try:
    f = open('zero_length_existing_file', 'rt')
    d = f.readline()
    print(len(d))
    f.close()
except IOError:
    print(-1)
```

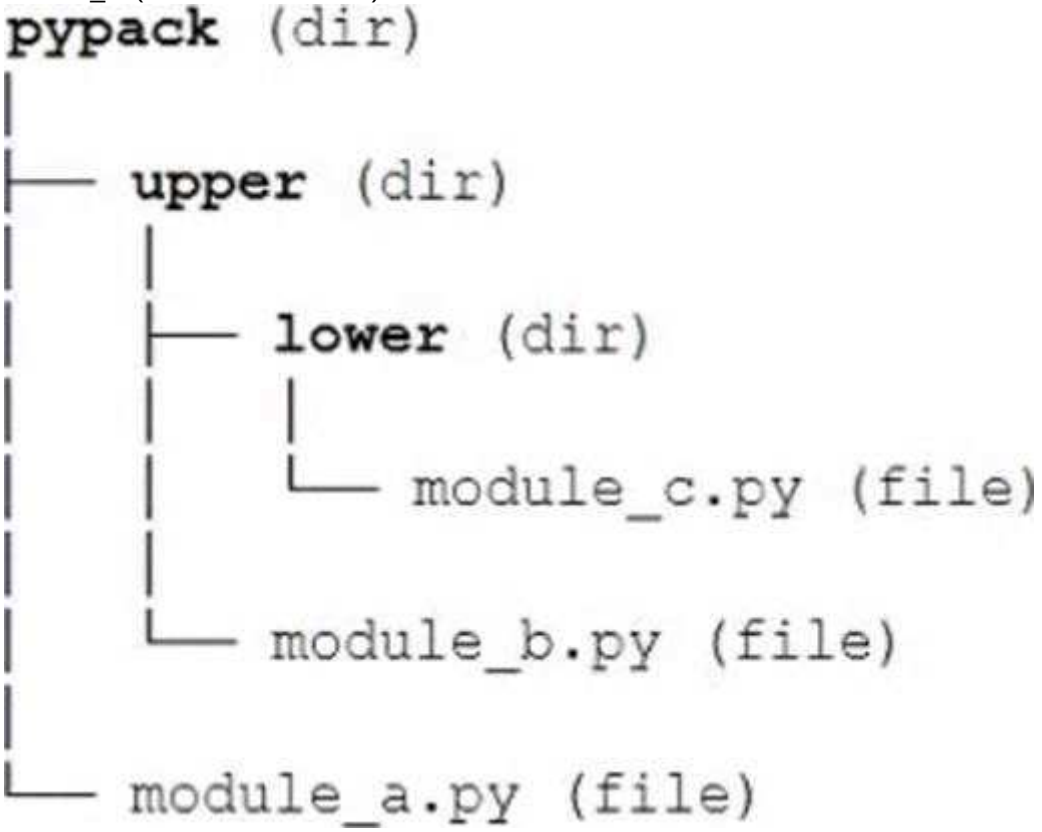
A. 0
B. -1
C. an errno value corresponding to file not found
D. 2

Correct Answer: A

Explanation

Explanation/Reference:
Since the file exists, no exception will occur even it is empty.
Even if there is no content, readline() will not raise any exception, it just returns an empty string "" which has a length of 0.

QUESTION 79
With regards to the directory structure below, select the proper forms of the directives in order to import module_a. (Select two answers)



A. `import pypack.module_a`
B. `import module_a from pypack`
C. `import module_a`
D. `from pypack import module_a`

Correct Answer: AD
Explanation

Explanation/Reference:

QUESTION 80
Which one of the platform module functions should be used to determine the underlying platform name?

A. platform.uname()
B. platform.platform()
C. platform.python_version()
D. platform.processor()

Correct Answer: B
Explanation

Explanation/Reference:
`uname()` shows platform general name e.g. "Windows", PC name, windows release e.g. "10" and windows version "10-10.0.17763", ... etc
`platform()` shows the platform name e.g. "Windows-10-10.0.17763-SP0"

Code for testing:

```
import platform
print(platform.uname())
print(platform.platform())
print(platform.python_version())
print(platform.processor())
```

QUESTION 81
What is a true about python class constructors? (Select two answers)

A. the constructor must have at least one parameter
B. the constructor must return a value other than `None`
C. the constructor is a method named `__init__`
D. there can be more than one constructor in a Python class.

Correct Answer: AC
Explanation

Explanation/Reference:
B is wrong since `__init__` must returns `None`.

QUESTION 82
Assuming that the following inheritance set is in force, which of the following classes are declared properly? (Select two answers)

```
class A:
    pass

class B(A):
    pass

class C(A):
    pass

class D(B):
    pass
```

A. `class Class_4(D, A): pass`

B. `class Class_1(C, D): pass`
C. `class Class_3(A, C): pass`
D. `class Class_2(B, D): pass`

Correct Answer: AB
Explanation

Explanation/Reference:
Within the SuperClass list, a child class must be specified before a parent class.

Code for testing.

```
class A:
~~~~pass

class B(A):
~~~~pass

class C(A):
~~~~pass

class D(B):
~~~~pass

class Class_4(D, A): pass
class Class_1(C, D): pass
# class Class_3(A, C): pass  # Error!! Parent A before Child C
# class Class_2(B, D): pass  # Error!! Parent B before Child D
```

QUESTION 83
What is the expected output of the following code?

```
def foo(x,y,z):
  return x(y) - x(z)

print{foo(lambda x: x % 2, 2, 1) )
```

A. 1
B. 0
C. -1
D. an exception is raised

Correct Answer: C
Explanation

Explanation/Reference:
In the print statement, the function "foo()" is called by passing 3 arugments:
- A function object created by lambda. The function returns the remainder after dividing by 2.
- The value 2.
- The value 1

In the function foo(), it calls the lambda function with the 2nd parameter i.e. 2 and then the 3rd parameter i.e. 1. Subtraction is performed on the two results returned. Since 2 % 2 = 0 and 1 % 2 = 1, the subtraction result is 0 - 1 = -1.

Code for testing:

```
def foo(x,y,z):
~~~~return x(y) - x(z)
print( foo(lambda x: x % 2, 2, 1) )
```

QUESTION 84

What is the expected output of the following code if existing_file is the name of a file located inside the working directory?

```
try:
    f = open('existing_file','w')
    print(1, end=' ')
except IOError as error:
    print(error.errno, end=' ')
    print(2, end=' ')
else:
    f.close()
    print(3, end=' ')
```

A. 1 2
B. 1 2 3
C. 1 3
D. 2 3

Correct Answer: C
Explanation

Explanation/Reference:
Since no exception occurs, only the print() statemnet in `try` and `else` clause are run.
Hence only 1 and 3 is printed to the console.

QUESTION 85
What is true about Python packages? (Select two answers)

A. the sys.path variable is a list of strings
B. _pycache_is a folder that stores semi-compiled Python modules
C. a package contents can be stored and distributed as an mp3 file
D. a code designed to initialize a package's state should be placed inside a file named init.py

Correct Answer: AB
Explanation

Explanation/Reference:
D is wrong since the file name must have two underscores before and after the word "init" i.e. "`__init.__.py`"

QUESTION 86
Which of the following statements are true? (Select two answers)

A. \e is an escape sequence used to mark the end of lines
B. ASCII is synonymous with UTF-8
C. II in ASCII stands for Information Interchange

D. a code point is a number assigned to a given character

Correct Answer: CD
Explanation

Explanation/Reference:
A is wrong since newline character i.e. \n mark the end of a line.
B is wrong since ASCII is just a subset of UTF-8

QUESTION 87
Which of the following lambda definitions are correct? (Select two answers)

A. `lanbda x,y: return x//y - x%y`
B. `lambda x,y: x//y - x%y`
C. `lambda (x,y) = x//y - x%y`
D. `lambda x,y: (x,y)`

Correct Answer: BD
Explanation

Explanation/Reference:
A is wrong since the keyword "return" forms a statement instead of an expression.
C is wrong since round braces are NOT allowed on the left side to enclose parameters.

QUESTION 88
What is the expected behavior of the following code?

```
my_list = [i for i in range(5, 0, -1)]
m = [my_list[i] for i in range(5) if my_list[i] % 2 == 0]
print(m)
```

A. the code is erroneus and it will not execute
B. it outputs [2, 4]
C. it outputs [4, 2]
D. it outputs [0, 1, 2, 3, 4]

Correct Answer: C
Explanation

Explanation/Reference:
`[i for i in range(5, 0, -1)]` produces [5, 4, 3, 2, 1]

Then:
`for i in range(5)` generates 0, 1, 2, 3, 4
When using as index in my_list, `my_list[i]` will returns 5, 4, 3, 2, 1

If `my_list[i]` is even number, it is returned.
Therefore 4 and 2 are returned as a new List object and assigned to `m`

Code for testing:

```
my_list = [i for i in range(5, 0, -1)]
m = [my_list[i] for i in range(5) if my_list[i] % 2 == 0]
print(m)        # [4, 2]
```

QUESTION 89
Which of the following lambda function definitions are correct? (Select two answers)

A. lambda X: None
B. lambda : 3.1415
C. lambda x: def fun(x): return x
D. lambda lambda: lambda * lambda

Correct Answer: AB
Explanation

Explanation/Reference:
C is wrong since "return ..." is a statement and statement is not allowed.
D is wrong since "lambda" is keyword / reserved word and cannot be used as parameter variable.

QUESTION 90
Assuming that the snippet below has been executed successfully, which of the following expressions will evaluate to True? (Select two answers)

```
string = 'python' [::2]
string = string[-1] + string[-2]
```

A. `string[0] == string[-1]`
B. `string is None`
C. `len(string) == 3`
D. `string[0] == 'o'`
E. `len(string) == 2`

Correct Answer: DE
Explanation

Explanation/Reference:
The original question found in Internet does not have choice E, it is added since there is only one correct answer among the first four choices.

Code for Testing

```
string = 'python' [::2]
print(string)               # pto
string = string[-1] + string[-2]
print(string)               # ot
print(string[0] == string[-1])  # False
print(string is None)           # False
print(len(string) == 3)         # False
print(string[0] == 'o')         # True
print(len(string) == 2)         # True
```

QUESTION 91
The following class definition is given.

```
Class Class:
    def ___init ___(self, val):
        self.val = val
    def get(self):
        return self.val
    def show(self):
        XXX
```

We want the show() method to invoke the get() method, and then output the value the get() method returns.

Which of the invocations should be used instead of XXX?

A. `print(get(self))`
B. `print(self.get())`
C. `print(get())`
D. `print(self.get(val))`

Correct Answer: B
Explanation

Explanation/Reference:
When calling another instance method, you should use `self.<method name>()`

QUESTION 92
A property that stores information about a given class's super-classes is named:

A. `__upper__`
B. `__bases__`
C. `__ancestors__`
D. `__super__`

Correct Answer: B
Explanation

Explanation/Reference:

QUESTION 93
What is the expected behavior of the following code?

```
my_list = [1, 2, 3]

try:
 my_list[3] = my_list[2]
except BaseException as error:
 print(error)
```

A. it outputs error
B. it outputs list assignment index out of range
C. the code is erroneous and it will not execute
D. it outputs <class ' IndexError' >

Correct Answer: B
Explanation

Explanation/Reference:
Since the largest index number of my_list is 2, "my_list[3]" will raise "IndexError" with the details "list assignment index out of range".

BaseException is the parent of all Exceptions and special exceptions. Therefore it can be used to catch "IndexError". Moreover, since "`__str__`" in built-in excpetion prints detailed message about an exception, therefore "list assignment index out of range" is printed..

Code for testing:

```
my_list = [1, 2, 3]
try:
~~~~my_list[3] = my_list[2]
except BaseException as error:
~~~~print(error)
```

QUESTION 94
Assuming that the following piece of code has been executed successfully, which of the expressions evaluate to True? (Select two answers)

```
class A:
    VarA = 1
    def __init__(self):
        self.prop_a = 1

class B(A):
    VarA = 2
    def __init__(self):
        self.prop_a = 2
        self.prop_aa = 2

class C(B):
    VarA = 3
    def __init__(self):
        super().__init__()

obj_a = A()
obj_b = B()
obj_c = C()
```

A. `obj_b.prop_a == 3`

B. `hasattr(obj_b, 'prop_aa')`
C. `isinstance(obj_c,A)`
D. `B.VarA == 3`

Correct Answer: BC
Explanation

Explanation/Reference:
A is wrong since Class B's `__init__` set the `prop_a` to 2.
D is wrong since Class B's class variable VarA is set to 2.

B is correct since Class B's `__init__` has defined the instance variable `prop_aa` for B's object.
C is correct since Class C's is a subclass of B which is a subclass of A. Since C is a indirectly subclass of A, C's object IS-A "A".

Code for testing:

```
class A:
~~~~VarA = 1
~~~~def __init__(self):
~~~~~~~~self.prop_a = 1

class B(A):
~~~~VarA = 2
~~~~def __init__(self):
~~~~~~~~self.prop_a = 2
~~~~~~~~self.prop_aa = 2

class C(B):
~~~~VarA = 3
~~~~def __init__(self):
~~~~~~~~super().__init__()

obj_a = A()
obj_b = B()
obj_c = C()

print(obj_b.prop_a == 3)            # False
print(hasattr(obj_b, 'prop_aa'))    # True
print(isinstance(obj_c,A))          # True
print(B.VarA == 3)                  # False
```

QUESTION 95
Which of the following statements are true? (Select two answers)

A. `open()` requires a second argument
B. `open()` is a function which returns an object that represents a physical file
C. instd, outstd, errstd are the names of pre-opened streams
D. if invoking `open()` fails, an exception is raised

Correct Answer: BD
Explanation

Explanation/Reference:
A is wrong since the 2nd argument can be omitted and the default mode is "rt".
C is wrong since the pre-opened streams have names `stdin`, `stdout` and `stderr`.

B is correct since open() is a built-in function, the object is actually a file stream object but you can say that the file stream object represents a physical file for reading and writing.

QUESTION 96

What is the expected output of the following snippet?

```
class Upper:
    def method(self):
        return 'upper'

class Lower(Upper):
    def method(self):
        return 'lower'

Object = Upper()
print(isinstance(Object,Lower), end=' ')
print(Object.method())
```

A. True lower
B. True upper
C. False upper
D. False lower

Correct Answer: C
Explanation

Explanation/Reference:
"Objec"t is an object of class Upper.
Since Upper is not subclass of Lower, `isinstaince(Object, Lower)` will return **`False`**
Since Object.method will call the method in class Upper and therefore "**`upper`**" is printed.

Code for testing:

```
class Upper:
~~~~def method(se1f):
~~~~~~~~return 'upper'

class Lower(Upper):
~~~~def method(self):
~~~~~~~~return 'lower'

Object = Upper()
print(isinstance(Object,Lower), end=' ')
print(Object.method())
```

QUESTION 97
Assuming that the code below has been placed inside a file named `code.py` and executed successfully, which of the following expressions evaluate to True? (Select two answers)

```
class ClassA:
    var = 1
    def __init__(self, prop):
        prop1 = prop2 = prop

class ClassB(ClassA):
    def __init__(self, prop):
        prop3 = prop ** 2
        super().__init__(prop)

Object = ClassB(2)
```

A. `str(Object) == 'Object'`
B. `__name__ == '__main__'`
C. `ClassA.__module__ == 'ClassA'`
D. `len(ClassB.__bases__) == 1`

Correct Answer: BD
Explanation

Explanation/Reference:
B is correct since the file is run directly instead of running as module
D is corrent since ClassB has only one direct SuperClass.

A is wrong since it str(Object) returns a String in the form "<class Upper object ....>"
C is wrong since ClassA is defined in a file that is being run directly (therefore ClassA.__module__ is "__main__")

Code for testing:

```
class ClassA:
~~~~var = 1
~~~~def __init__(self, prop):
~~~~~~~~prop1 = prop2 = prop

class ClassB(ClassA):
~~~~var = 1
~~~~def __init__(self, prop):
~~~~~~~~prop3 = prop ** 2
~~~~~~~~super().__init__(prop)

Object = ClassB(2)
print(str(Object) == 'Object')          # False
print(__name__ == '__main__')           # True
print(ClassA.__module__ == 'ClassA')    # False
print(len(ClassB.__bases__) == 1)       # True
```

```
print(vars(Object))                    # {}
# There is no instance variable since all variables defined in __init__ are
without qualifier self.
```

QUESTION 98
You are going to read 16 bytes from a binary file into a bytearray called data.

Which lines would you use? (Select two answers)

A. data = bytearray(16)
   bf.readinto(data)
B. data = bf.read(bytearray(16))
C. bf. readinto(data = bytearray(16))
D. data = bytearray(bf.read(16))

Correct Answer: AD
Explanation

Explanation/Reference:
Assume bf is:
`bf=open("somefile","rb")`

A and D is the proper way to read byte data into a bytearray object.

QUESTION 99
Which of the following lines of code will work flawlessly when put independently inside the dup() method in order to make the snippet's output equal to [0, 1, 1]? (Select two answers)

```
class MyClass:
    def __init__(self, initial):
        self.store = initial

    def put(self, new):
        self.store.append(new)

    def get(self):
        return self.store

    def dup(self):
        # insert the line of code here

Object = MyClass([0])
Object.put(1)
Object.dup()
print(Object.get())
```

A. `put(self.store[1])`
B. `self.put(store[1])`
C. `self.put(self.get()[-1])`
D. `self.put(self.store[1])`

Correct Answer: CD
Explanation

Explanation/Reference:
The list object `[0]` is passed to "`__init__`" when MyClass' object "Object" is created. It is assigned to the Instance variable `store`.
Then `put(1)` is call. The value 1 is assigned to the parameter variable "new" and is used as argument to append to the instance vairbale `store`. Hence store now refer to the List object `[0, 1]`

Something need to perform to make the List object becomes `[0, 1, 1]` so that the `get()` method will return it.

A is wrong since you are just modifying the 2nd element in the List object "store".
B is wrong since you need a object reference to access the Instance variable "`store`".

C is correct since `self.get()` will return `[0,1]`. The element of the index -1 is 1. Through `self.put()`, the 1 can be added to the List object "`store`" to become `[0, 1, 1]`
D is correct since `self.store` returns `[0,1]`. The element of the index 1 is 1. Through `self.put()`, the 1 can be added to the List object "`store`" to become `[0, 1, 1]`

Code for testing:

```
class MyClass:
~~~~def __init__(self, initial):
~~~~~~~~self.store = initial

~~~~def put(self, new):
~~~~~~~~self.store.append(new)

~~~~def get(self):
~~~~~~~~return self.store

~~~~def dup(self):
~~~~~~~~# put(self.store[1])         # Error
~~~~~~~~# self.put(store[1])         # Error
~~~~~~~~self.put(self.get()[-1])   # Correct. choice C
~~~~~~~~# self.put(self.store[1])  # Correct. choice D

Object = MyClass([0])
Object.put(1)
Object.dup()
print(Object.get())
```

You can uncomment one single line in the method dup() each time to test the choices

QUESTION 100
What is true about Python packages? (Select two answers)

A. the__name__variable content determines the way in which the module was run
B. a package can be stored as a tree of sub-directories/sub-folders
C. __pycache__ is the name of a built-in variable
D. hashbang is the name of a built-in Python function

Correct Answer: AB
Explanation

Explanation/Reference:

QUESTION 101
What is the expected behavior of the following code?

```
m = 0

def foo(n):
    global m
    assert m != 0
    try:
        return 1/n
    except ArithmeticError:
        raise ValueError

try:
    foo(0)
except ArithmeticError:
    m += 2
except:
    m += 1

print(m)
```

A. it outputs 2
B. the code is erroneous and it will not execute
C. it outputs 1
D. it outputs 3

Correct Answer: C
Explanation

Explanation/Reference:
When the function "`foo()`" is called, the variable "m" is defined to use that of the global variable i.e. with value 0.
Then `assert` want to make sure m is not 0. Since the condition is False, AssertionError is therefore raised.

The AssertionError is raise outside of the try statement within the function and therefore you do not need to check if that try statement can handle it.
Then returning to the calling statement, since foo() is called in a try clause, the except clauses are checked.
- AsserionError cannot be handled by ArithmeticError.

- AssertionError can be handled by default BaseException since BaseException is the SuperClass of AssertionError. Hence m += 1 is run.
The value of m therefore becomes 0 + 1 = 1

Code for testing:

```
m = 0

def foo(n):
~~~~global m
~~~~assert m != 0
~~~~try:
~~~~~~~~return 1/n
~~~~except ArithmeticError:
~~~~~~~~raise ValueError

try:
~~~~foo(0)
except ArithmeticError:
~~~~m  += 2
except:
~~~~m += 1

print(m)
```

QUESTION 102
A Python module named pymod.py contains a variable named pyvar.

Which of the following snippets will let you access the variable? (Select two answers)

A.
```
import pyvar from pymod
pyvar = 1
```
B.
```
from pymod import *
pyvar = 1
```
C.
```
from pymod import pyvar
pyvar()
```
D.
```
import pymod
pymod.pyvar = 1
```

Correct Answer: BD
Explanation

Explanation/Reference:
A is wrong since `from` must be entered before `import`
C is wrong since pyvar is NOT a function and cannot be called with pyvar().

Remarks :
Note that in B, the value 1 is actually assigned to the "pyvar" imported to the main program. The value of the "pyvar" in the module remains to be its initial value.

QUESTION 103
Which is a true about python class constructors? (Select two answers)

A. can return a value
B. can be invoked directly from inside the class
C. can be invoked directly from any of the subclasses
D. can be invoked directly from any of the superclasses

Correct Answer: BC
Explanation

Explanation/Reference:
Since constructor in this syllabus refers to __init__, it can only return None.
Since a superclass has no reference to the subclass, it cannot call the class constructor directly.

B is correct. An instance method of the same class can call "`self.__init()`".
C is correct. A Subclass can call "`super().__init__()`".

QUESTION 104
What is the expected output of the following code of the file named zero_length_existing_file is a zero-length file located inside the working directory?

```
try:
    f = open('zero_length_existing_file', 'rt')
    d = f.readline()
    print(len(d))
    f.close()
except IOError:
    print(-1)
```

A. the length of the first line from the file
B. -1
C. the number of lines contained inside the file
D. the length of the last line from the file

Correct Answer: A
Explanation

Explanation/Reference:
Same program code as Q78 but the choices are different.

QUESTION 105
Which of the following expression evaluate to True? (Select two answers)

A. `len('\'') == 1`
B. `len("""
"""") == 0`
C. `chr(ord('A') + 1) == 'B'`
D. `ord("Z") - ord("z") == ord("0")`

Correct Answer: AC
Explanation

Explanation/Reference:
A is correct since the string contains only 1 character '
C is corrent since the character B is the character just after A in ASCII code.

B is wrong since the string contains a newline character
D is wrong since the result of `ord("Z") - ord("z")` is -32 (since "Z" is before i.e. smaller than "z") but `ord()` for "0" or any other character will always return a positive result.

Code for testing:

```
print(len('\'') == 1)  # True
print(len("""
""") == 0)  # False
```

```
print(chr(ord('A') + 1) == 'B')  # True
print(ord("Z") - ord("z") == ord("0"))  # False
```

QUESTION 106
Assuming that the following code has been executed successfully, select the expressions which evaluate to True (Select three answers)

```
var = 1

def f():
    global var
    var += 1
    def g():
        return var
    return g

a = f()
b = f()
```

A. `a is b`
B. `b() > 2`
C. `a() > 2`
D. `a is not None`

Correct Answer: BCD
Explanation

Explanation/Reference:
For the above, unlike the normal use of nested function, no Local variable is involved. All "`var`" is refering to the Global variable.
Hence the nested function object will always return a value based on the current value of the Global variables.

Note that although a and b stores nested function object which has no local variable and do exactly the same thing.
Since `f()` is called independantly each time, two different nested function objects are returned.

For the Global variable "var":
- It is initially set to 1.
- When f() is called the first time, it is incremented by 1 to 2.
- When f() is called the second time, it is incremented by 1 to 3.

Hence if you call `a()` and `b()`, both will return 3.

Code for testing

```
var = 1

def f():
~~~~global var
~~~~var += 1
~~~~def g():
~~~~~~~~return var
~~~~return g

a = f()
b = f()

print(a())              # prints 3
print(b())              # prints 3

print(a is b)           # False
print(b() > 2)          # True
print(a() > 2)          # True
print(a is not None)    # True

var = 99
print(a())              # prints 99
print(b())              # prints 99
```

Special Noltes about this question:
This question from Internet actually asks to "Select two answer". However since there are actually three correct answer, the question here is modified "Select three".

QUESTION 107
What is true about Python packages? (Select two answers)

A. a package is a group of related modules
B. the pyc extension is used to mark semi-compiled Python packages
C. the __name__ variable always contains the name of a package
D. a package is a single file whose name ends with the pa extension

Correct Answer: AB
Explanation

Explanation/Reference:
C is wong since `__name__` can return:
- `__main__` when a module file in the package is run directly.
- the module name (NOT the pacakge name) when a module file within the package is loaded as a module.

QUESTION 108
Assuming that the code below has been executed successfully, which of the following expressions will always evaluate to True (Select
two answers)

```
import random

random.seed(1)
v1 = random.random( )
random.seed(1)
v2 = random.random()
```

A. `v1 >= 1`
B. `v1 == v2`
C. `len(random.sample([1, 2, 3],2)) > 2`
D. `random.choice([1, 2, 3]) >= 1`

Correct Answer: BD
Explanation

Explanation/Reference:
A is wrong since random() generates a value < 1.
C is wrong since sample() here generates two elements and the `len()` is always 2 (i.e. NOT >2)

B is correct since after generating a new seed of a fixed value i.e. 1, the first random number will alrways be the same.
D is correct since choice() here must return either 1, 2 or 3 therefore the returned value must be >= 1

Code for testing:

```
import random

random.seed(1)
v1 = random.random()
random.seed(1)
v2 = random.random()

print(v1 >= 1)                              # False
print(v1 == v2)                             # True
print(len(random.sample([1,2,3],2)) > 2)  # False
print(random.choice([1,2,3]) >= 1)        # True
```

QUESTION 109
Which one of the platform module functions should be used to determine the undertying OS version?

A. `platfom.python_version()`
B. `platform.python_version_tuple()`
C. `platform.processor()`
D. `platform.version()`

Correct Answer: D
Explanation

Explanation/Reference:

QUESTION 110
What is the expected output of the following code?

```
import sys

b1 = type(dir(sys)) is str
b2 = type(sys.path[-1]) is str
print(b1 and b2)
```

A. False
B. None
C. 0
D. True

Correct Answer: A
Explanation

Explanation/Reference:
`dir(sys)` returns a List object of Strings (representing attribute names of the sys module). Hence b1 is False.
`sys.path` returns a List object containing search folders for modules. `sys.path[-1]` (i.e. last element of

`sys.path`) returns a string representing a folder. Hence b2 is True.

Since `"False and True"` returns `False`, A is the answer.

Code for testing:

```
import sys

b1 = type(dir(sys)) is str
b2 = type(sys.path[-1]) is str
print(b1 and b2)
print()

print("For dir(sys) :")
print(type(dir(sys)))
print(dir(sys))
print()

print("For sys.path[-1] :")
print(type(sys.path[-1]))
print(sys.path[-1])
print()

print("For sys.path :")
print(type(sys.path))
print(sys.path)
print()
```

QUESTION 111
A Python module named pymod.py contains a function named `pyfun()`.

Which of the following snippets will let you invoke the function? (Select two answers)

A.
```
import pyfun from pymod
pyfun()
```
B.
```
from pymod import pyfun
pyfun()
```
C.
```
from pymod import *
pymod.pyfun()
```
D.
```
import pymod
pymod.pyfun()
```

Correct Answer: BD
Explanation

Explanation/Reference:
A is wrong since `from` must be before `import`
C is wrong since you should NOT include the module name as qualifier when you are importing all attributes from a module using "`from ... import ...`"

QUESTION 112
What is the expected behavior of the following code?

```
s='2A'

try:
    n = int(s)
except TypeError:
    n = 3
except LookupError:
    n = 2
except:
```

```
    n = 1
print(n)
```

A. the code is erroneous and it will not execute
B. it outputs 2
C. it outputs 1
D. it outputs 3

Correct Answer: C
Explanation

Explanation/Reference:
The string "2A" cannot be converted into "int" and therefore a ValueError is raised.
Since ValueError cannot be handled by "TypeError" and "LookupError", it can only be handled by the default "BaseException" of the last `except` clause.
Therefore n will be assigned with 1.

Remarks :
Unlike other similar questions, no error will occur since the `except` clause using the default BaseException is the last `except` clause in the `try` statement.

Code for testing:

```
s = '2A'

try:
~~~~n = int(s)
except TypeError:
~~~~n = 3
except LookupError:
~~~~n = 2
except:
~~~~n = 1

print(n)
```

QUESTION 113
Which of the following expressions evaluate to True? (Select two answers)

A. `3 * 'a' < 'a' * 2`
B. `'1'+'2' * 2 != '12'`
C. `'abc'.upper() < 'abc'`
D. `11 == '011'`

Correct Answer: BC
Explanation

Explanation/Reference:
A is wrong since shorter string "aa" is smaller than longer string "aaa"
D is wrong since False will always be returned for two different types. The left is a `int` and the right is a `str`.

B is correct. Note that "`*`" is performe first, the result of the left side of "=" is `'1'+'22'` i.e. `'122'`
C is correct. Note that "`'abc'.upper()`" is `'ABC'` and uppercase "A" is smaller than the lowercase "a".

Code for testing:

```
print(3 * 'a' < 'a' * 2)        # False
print('1'+'2' * 2 != '12')      # True
```

```
print('abc'.upper() < 'abc')    # True
print(11 == '011')              # False
```

QUESTION 114
Which of the following invocations are valid'? (Select two answers)

A. `"python".find("")`
B. `'python'.sorted()`
C. `sort("python")`
D. `"python".rindex("th")`

Correct Answer: AD
Explanation

Explanation/Reference:
B is wrong since `sorted()` is a built-in function and cannot be called by using a String as qualifier.
C is wrong since `sort()` is not a built-in function.

Note that for A, the result of finding an empty string in a String will always return 0.

Code for testing:

```
print("Python".find(""))        # 0
# print('python'.sorted())      # Error
# print(sort("python"))         # Error
print("python".rindex("th"))    # 2
```

QUESTION 115
What is the expected behavior of the following code?

```
string = '123'
dummy = 0
for character in reversed(string):
    dummy += int(character)
print(dummy)
```

A. it outputs 321
B. it outputs 6
C. it raises an exception
D. it outputs 123

Correct Answer: B
Explanation

Explanation/Reference:
`reversed(string)` will return a `reversed` object that can yield the characters "3", "2" and "1".
By iterating those characteres and converting each of them to `int` value to obtain a sum, 6 will be obtained.

Code for testing:

```
string = '123'
dummy = 0
for character in reversed(string):
~~~~dummy += int(character)
print(dummy)
```

QUESTION 116
Assuming that the snippet below has been executed successfully, which of the following expressions will evaluate to True? (Select

two answers)

```
string = 'SKY'[::-1]
string = string[-1]
```

A. `string` is None
B. `string[0] == 'Y'`
C. `len(string) == 1`
D. `string[0] == string[-1]`

Correct Answer: CD
Explanation

Explanation/Reference:
The String "`string`" having 'SKY' create a new String "YKS" (i.e. reversed) after the Slice notation `[::-1]` and then it is assigned back to "`string`".
Since [-1] i.e. the last character of above reserved "`string`" is "S", this character is assigned back to "`string`"

C is correct since there is only one character in "`string`"
D is correct since the first character is the same as the last character when there is only one character in "`string`"

Code for testing:

```
string = 'SKY'[::-1]
string = string[-1]
print(string)          # S
print(string is None)          # False
print(string[0] == 'Y')        # False
print(len(string) == 1)        # True
print(string[0] == string[-1])  # True
```

QUESTION 117
What is the expected behavior of the following code?

```
the_list = "alpha;beta:gamma".split(":")
the_string = ''.join(the_list)
print(the_string.isalpha())
```

A. it outputs nothing
B. it raises an exception
C. it outputs True
D. it outputs False

Correct Answer: D
Explanation

Explanation/Reference:
When splitting with "`:`", since the words are separated by both semicolon "`;`" and colon "`:`", only 2 Substrings is formed and the List contains two elements "`alpha;beta`" and "`gama`"
After joining, the result string will be "`alpha;betagamma`".
Since there is a semicolon in the String, `isalpha()` which checks for alphabet returns `False`

Remarks :
If the original string contains colon "`:`" only (i.e. semicolon is changed to colon"), the answer will be `True`.

Code for testing:

```
the_list = "alpha;beta:gamma".split(":")
print(the_list)                 # ['alpha;beta', 'gamma']
the_string = ''.join(the_list)
print(the_string)               # alpha;betagamma
print(the_string.isalpha())     # False

# if semicolon is changed to colon in the question
the_list2 = "alpha:beta:gamma".split(":")
print(the_list2)                 # ['alpha', 'beta', 'gamma']
the_string2 = ''.join(the_list2)
print(the_string2)               # alphabetagamma
print(the_string2.isalpha())     # True
```

QUESTION 118
Which of the following expressions evaluate to True? (Select two answers)

A. `'dcb'` not in `'abcde'[::-1]`
B. `str(1-1)` in `'0123456789'[:2]`
C. `'phd'` in `'alpha'`
D. `'True'` not in `'False'`

Correct Answer: BD
Explanation

Explanation/Reference:
A is wrong. `'abcde'[::-1]` will become `'edcba'` which contains `'dcb'`. Hence "`not in`" returns `False`.
C is wrong. `'phd'` cannot be found in `'alpha'`;

B is correct. 1-1 becomes 0 and creates the string `'0'`. This `'0'` is included in `'02468'` (which is obtained from `'0123456789'[:2]`)

D is correct. `'True'` cannot be found in `'False'` therefore "`not in`" returns `True`.

Code for testing:

```
print('dcb' not in 'abcde'[::-1])     # False
print(str(1-1) in '0123456789'[:2])   # True
print('phd' in 'alpha')               # False
print('True' not in 'False')          # True
```

QUESTION 119
What is the expected behavior of the following code?

```
class Class:
    Var = 0
    def __foo(self):
        Class.Var += 1
        return Class.Var

o = Class()
o._Class__foo()
print(o._Class__foo())
```

A. It raises an exception
B. it outputs 1
C. it outputs 2
D. it outputs 3

Correct Answer: C
Explanation

Explanation/Reference:
Since the instance method foo has two underscore before it, it is name mangled to `_Class__foo()`

In the class Class, the class variable "`Var`" is assigned with a initial value of 0.
When the method `_Class__foo()` called, the class variable "`Var`" is incremented by 1 and its value is returned (but the returned value is not used).
Since `_Class__foo()` called again within print, the class variable "`Var`" is incremented by 1 again to 2.
This time the returned value 2 will be output be print..

Code for testing:

```
class Class:
~~~~Var = 0
~~~~def __foo(self):
~~~~~~~~Class.Var += 1
~~~~~~~~return Class.Var

o = Class()
o._Class__foo()
print(o._Class__foo())
```

QUESTION 120
Which of the following lines of code will work flawlessly when put independently inside the `inc()` method in order to make the snippet's output equal to 3? (Select two answers)

```
class MyClass:
    Var = 0
    def __init__(self):
        MyClass.Var += 1
        self.prop = MyClass.Var

    def get(self):
        return self.prop

    def put(self, val):
        self.prop = val

    def inc(self, val):
        # insert the line of code here

Object = MyClass()
Object.inc(2)
print(Object.get())
```

A. `self.put(self.prop + val)`
B. `self.put(self.get() + val)`
C. `self.put(get() + val)`
D. `put(self.prop + val)`

Correct Answer: AB
Explanation

Explanation/Reference:
`Var` is a class variable initialized to 0
When the object `Object` is created, __init__ will increment Var by 1 i.e. become 1. Then this value 1 is assigned to the Instance variable `prop`.
Therefor `prop` has a value of 1.

The last statement `Object.get()` will obtain the Instance variable `prop`

Since the value of prop in the last statement must be 3, `Object.inc(2)` should add `2 to prop`.

Actually, you can find the right answer just by checking whether the instance method is called properly:
C is wrong since `get()` must be called with object reference i.e. "`self.get()`"
D is wrong since `put()` must be called with object reference i.e. "`self.put()`".

Since both `self.prop` and `self.get()` obtain the value 1 from `prop`. In A and B, both added the value of `prop` with the parameter variable having value 2 (passed by arugment) to obtain a value 3 for using `self.put()` to set the value 3 to `prop`.

Code for testing:

```
class MyClass:
~~~~Var = 0
~~~~def __init__(self):
~~~~~~~~MyClass.Var += 1
~~~~~~~~self.prop = MyClass.Var

~~~~def get(self):
~~~~~~~~return self.prop

~~~~def put(self, val):
~~~~~~~~self.prop = val

~~~~def inc(self, val):
~~~~~~~~self.put(self.prop + val)   # choice A
~~~~~~~~# self.put(self.get() + val)  # choice B

Object = MyClass()
Object.inc(2)
print(Object.get())
```

QUESTION 121
What is the expected behavior of the following code?

```
class Class:
    Variable = 0
    def __init__ (self):
        self.value = 0

object_1 = Class()
object_1.Variable += 1
object_2 = Class()
object_2.value += 1
print(object_2.Variable + object_1.value)
```

A. it outputs 2
B. it outputs 0
C. it outputs 1
D. it raises an exception

Correct Answer: B
Explanation

Explanation/Reference:
The class variable "`Variable`" is initialized to 0
All objects will have an Instance variable "`value`" with initial value 0.

The 2nd statement add the value of the class variable "Variable" i.e. 0 with 1 but the result is assigned to a newly created instance variable "Variable" in object_1. Therefore class variable "Variable" remains 0.

The 4th statement add object_2's instance variable "value" by 1 and assign the result value 1 back to the

instance variable "value" in object_2.

Since class variable "Variable" remains 0 and instance variable "value" of object_1 remains 0. The sum printed is 0.

Code for testing:

```
class Class:
~~~~Variable = 0
~~~~def __init__ (self):
~~~~~~~~self.value = 0

object_1 = Class()
object_1.Variable += 1
object_2 = Class()
object_2.value += 1
print(object_2.Variable + object_1.value)
print()
print(Class.Variable)           # 0
print(object_1.__dict__)        # {'value': 0, 'Variable': 1}
print(object_2.__dict__)        # {'value': 1}
```

QUESTION 122
What is true about the __bases__ property?

A. there is no such property
B. it is accessible inside a class
C. it is accessible inside an object
D. it is accessible inside a class and an object

Correct Answer: B
Explanation

Explanation/Reference:
It is an attribue from `type` and therefore can only be accessed by class.

Remarks:
Actually the wordings should be:
It is accessible by using a class name as qualifier. You can access it within or outside the class definition with the qualifier.

QUESTION 123
What is the expected output of the following snippet?

```
class Upper:
    def method(self):
        return 'upper'

class Lower(Upper):
    def method(self):
        return 'lower'

Object = Lower()
print(isinstance(Object,Upper), end=' ')
print(Object.method())
```

A. True lower
B. True upper
C. False upper
D. False lower

Correct Answer: A
Explanation

Explanation/Reference:
Similar to Q96 but the object and the 2nd argument supplied to the built-in function `isinstance()` are different.

"Object" is an object of class Lower.
Since Lower is a subclass of Upper, `isinstaince(Object, Upper)` will return **True**
Since Object.method will call the method in class Lower and therefore "**`lower`**" is printed.

Code for testing:

```
class Upper:
~~~~def method(se1f):
~~~~~~~~return 'upper'

class Lower(Upper):
~~~~def method(self):
~~~~~~~~return 'lower'

Object = Lower()
print(isinstance(Object,Upper), end=' ')
print(Object.method())
```

QUESTION 124
Assuming that the code below has been placed inside a file named code.py and executed successfully, which of the following expressions evaluate to True? (Select two answers)

```
class ClassA:
    var = 1
    def __init__(self, prop):
        propl = prop2 = prop
    def __str___(self):
        return 'Object'

class ClassB(ClassA):
    def __init__(self, prop):
        prop3 = prop ** 2
        super().__init__(prop)

Object = ClassB(2)
```

A. `__name__ == 'code-py'`
B. `ClassA.__module__ == '__main__'`
C. `len(ClassB.__bases__) == 2`
D. `str(Object) == 'Object'`

Correct Answer: BD
Explanation

Explanation/Reference:
When "Object" is created, it calls the ClassB's contructor with a value of 2 as agrument:
- This assign square of the arugment 2 passed i.e. 4 to a local variable "prop3" (since it has no qualifier)
- It calls ClassA's constructor with the value 2

- In ClassA's constructor, two local variables "prop" and "prop2" are defined and initialized to the argument value 2 passed.

A is wrong since when the file is run directly, "__name__" returns "__main__".
B is correct since ClassA is defined in the file which is being run directly.
C is wrong since ClassB has only one direct SuperClass and therefore there is only one element in "ClassB.__bases__"
D is correct. It seems strange but since the __str__() method in ClassA is inherited by ClassB. This method is call when str() is called with "Object". The method returns the fixed string 'Object'.

Code for testing:

```
class ClassA:
~~~~var = 1
~~~~def __init__(self, prop):
~~~~~~~~propl = prop2 = prop
~~~~def __str__(self):
~~~~~~~~return 'Object'

class ClassB(ClassA):
~~~~def __init__(self, prop):
~~~~~~~~prop3 = prop ** 2
~~~~~~~~super().__init__(prop)

Object = ClassB(2)

print(__name__ == 'code-py')             # False
print(ClassA.__module__ == '__main__')  # True
print(len(ClassB.__bases__) == 2)       # False
print(str(Object) == 'Object')          # True
```

QUESTION 125
Assuming that the code below has been executed successfully, which of the following expressions evaluate to True? (Select two answers)

```
class Class:
    var = data = 1
    def __init__(self, value):
        self.prop = value

Object = Class(2)
```

A. `'data' in Class.__dict__`
B. `len(Class.__dict__) == 1`
C. `'data' in Object.__dict__`
D. `'var' in Class.__dict__`

Correct Answer: AD
Explanation

Explanation/Reference:
A returns `True` since "data" is defined as Class variable in the class "Class"
B returns False since Class.__dict__ contains a lot of things (including default entries such as "__module__", "__dict__" ... etc) and will NOT have just one item
C returns False since "data" is not an Instance variable and therefore it cannot be found in Object.__dict__
D returns `True` since "var" is also defined as Class variable.

Code for testing:

```
class Class:
~~~~var = data = 1
~~~~def __init__(self, value):
```

```
~~~~~~~~self.prop = value

Object = Class(2)

print('data' in Class.__dict__)   # True
print(len(Class.__dict__) == 1)   # False
print('data' in Object.__dict__)  # False
print('var' in Class.__dict__)    # True
```

QUESTION 126
Assuming that the following inheritance set is in force, which of the following classes are declared properly? (Select two answers)

```
class A:
    pass

class B(A):
    pass

class C(A):
    pass

class D(B, C):
    pass
```

A. `class Class_4(C,B): pass`
B. `class Class_1(D): pass`
C. `class Class_3(A,C): pass`
D. `class Class_2(A,B): pass`

Correct Answer: AB
Explanation

Explanation/Reference:
A is correct regardless of order since C and B has no parent / child relationship
B is correct
C is wrong since the 1st class A is a parent of the 2nd class C. It is not allowed.
D is wrong since the 1st class A is a parent of the 2nd class B. It is not allowed.

QUESTION 127
What is true about Object-Oriented Programming in Python? (Select two answers)

A. a class is a recipe for an object
B. encapsulation allows you to hide a whole class inside a package
C. the arrows on a class diagram are always directed from a superclass towards its subclass
D. each object of the same class can have a different set of properties

Correct Answer: AD
Explanation

Explanation/Reference:
Remarks :
Encapsulation being implemented in Python is a form of weak encapsulation. It is just a convention telling you that a specific attribute with "_" before its name should not be accessed directly but no hiding / access control is implemented to prevent you from accessing it.

QUESTION 128
What is the expected behavior of the following code?

```
class Super:
    def make(self):
        pass
    def doit(self):
        return self.make()

class Sub_A(Super):
    def make(self):
        return 1

class Sub_B(Super):
    pass

a = Sub_A()
b = Sub_B()
print(a.doit() + b.doit())
```

A. it raises an exception
B. it outputs 2
C. it outputs 1
D. it outputs 0

Correct Answer: A
Explanation

Explanation/Reference:
Since class Sub_A has no `dotit()` method defined, `a.dotit()` will call the inherited method `doit()` defined in class Super. It then calls Instance method `make()` but since a is a Sub_A object, the `make()` in class SubA will be called. Hence 1 is return.

Since class Sub_B has no `dotit()` method defined, `b.dotit()` will call the inherited method `doit()` defined in class Super. It then calls Instance method `make()`. But siince Sub_B has no `make()` defined, the inherited `make()` in class Super will be called. Hence `None` is return.

However, since NoneType cannot add with int value 1, error will occur.

Code for testing:

```
class Super:
~~~~def make(self):
~~~~~~~~pass
~~~~def doit(self):
~~~~~~~~return self.make()

class Sub_A(Super):
~~~~def make(self):
~~~~~~~~return 1

class Sub_B(Super):
~~~~pass

a = Sub_A()
b = Sub_B()
print(a.doit() + b.doit())
```

QUESTION 129
Assuming that the following code has been executed successfully, select the expressions which evaluate to True (Select two answers)

```
def f(x,y):
    nom, denom = x, y
    def g():
        return nom / denom
```

```
    return g

a = f(1,2)
b = f(3,4)
```

A. `b() == 4`
B. `a != b`
C. `a is not None`
D. `a() == 4`

Correct Answer: BC
Explanation

Explanation/Reference:
The function `f()` returns the nested function object with local variables nom and denom set to different values supplied as arugments.
Hence a and b will be assignd with different nested function object having different values for the local variables.

`a()` has nom = 1 and denom = 2 and therefore will return 1/2 i.e. 0.5
`b()` has nom = 2 and denom = 4 and therefore return 3/4 i.e. 0.74

A is wrong
B is correct
C is correct
D is wrong

Code for testing:

```
def f(x,y):
~~~~nom, denom = x, y
~~~~def g():
~~~~~~~~return nom / denom
~~~~return g

a = f(1,2)
b = f(3,4)

print(a())   # prints the result of 1/2
print(b())   # prints the result of 3/4

print(b() == 4)        # False
print(a != b)          # True
print(a is not None)   # True
print(a() == 4)        # False
```

QUESTION 130
What is the expected out of the following code of the file named existing_text_file is a non-zero length file located inside the working directory?

```
try:
    f = open('existing_text_file', 'rt')
    d = f.readlines()
    print(len(d))
    f.close()
except IOError:
    print(-1)
```

A. the length of the first line from the file
B. -1
C. the number of lines contained inside the file

D. the length of the last line from the file

Correct Answer: C
Explanation

Explanation/Reference:
Looks similar to Q104 but:
- this is reading a non-zero length file
- readlines() is used instead of readline().

Since readlines() return a List having lines as elements for assigning to "d", len(d) is the number of elements in the List i.e number of lines in the file.

QUESTION 131
What is the expected behavior of the following code?

```
my_list = [i for i in range(5)]
m = [my_list[i] for i in range(4, 0, -1) if my_list[i] % 2 != 0]
print(m)
```

A. the code is erroneus and it will not execute
B. it outputs `[1, 3]`
C. it outputs `[4, 2, 0]`
D. it outputs `[3, 1]`

Correct Answer: D
Explanation

Explanation/Reference:
`[i for i in range(5)]` generates the List object [0, 1, 2, 3, 4]

Then
- since `for i in range(4, 0, -1)` generates 4, 3, 2, 1
- each of the above value being iterated is used an index in `my_list[i]` and will return the elements 4, 3, 2, 1 (e.g. mylist[4] is 4)
- `if my_list[i] % 2 != 0]` will returns True if the element in the List is an odd number.
Therefore only odd number is returned i.e. 3, 1.

Code for testing:

```
my_list = [i for i in range(5)]
m = [my_list[i] for i in range(4, 0, -1) if my_list[i] % 2 != 0]
print(m)   # [3, 1]
```

QUESTION 132
What is the expected behavior of the following code?

```
x = 3 % 1
y = 1 if x > 0 else 0
print(y)
```

A. it outputs 0
B. it outputs 1
C. the code is erroneus and it will not execute
D. it outputs -1

Correct Answer: A
Explanation

Explanation/Reference:
The remember of 3 / 1 is 0 (all number are divisible by 1) and is assigned to `x`.

Since `x > 0` is `False`, the if ... else ... expression returns the second value i.e. `0` for assigning to `y`.

Hence, printing "y" will show "0".

You can copy and paste the code in the question directly for testing.

QUESTION 133
Which of the following statements are true? (Select one answer)

A. the second `open()` argument is optional
B. `open()` is a function which returns an `int` that represents a physical file handle
C. instd, outstd, errstd are the names of pre-opened streams
D. if invoking `open()` fails, the value `None` is returned

Correct Answer: A
Explanation

Explanation/Reference:
B is wrong since it returns a file stream object, not an `int` value.
C is wrong since the names are stdin, stdout and stderr.
D is wrong since exception occurs and the statement is skipped. Then either the program aborts or jump to the except clause.

Special Noltes about this question:
This question from Internet actually asks to "Select two answer". However since there are actually one correct answer, the question here is modified "Select one".

QUESTION 134
What is the expected output of the following code?

```
myli = range(-2,2)
m = list(filter(1ambda x: True if abs(x) < 1 else False, myli))
print(len(m))
```

A. 16
B. 4
C. an exception is raised
D. 1

Correct Answer: D
Explanation

Explanation/Reference:
A `range` object "`myli`" is formed using List literal having elements -2, -1, 0, 1
`filter()` uses a Lambda function which returns True when absolute value of the parameter is smaller than 1 (i.e. only 0 will return True) and returns False otherwise. This function is used for filtering the 2nd argument i.e. the `range` object `myli`

A new list is formed from the `filter` object having the single element `0` and assigned to "`m`"
Hence the number of elements in the `lst` is now 1.

Remarks : For `abs(-1)`, the result is 1. Hence only 0 will returns True for the Lambda function..

Code for testing:

```
myli = range(-2,2)
m = list(filter(lambda x: True if abs(x) < 1 else False, myli))
print(len(m))
```

QUESTION 135
What is the expected output of the following code?

```
def foo(x,y,z):
  return x(y(z))

print( foo(lambda x: 2*x, lambda x: x//2, 2))
```

A. 2
B. 3
C. 4
D. an exception is raised

Correct Answer: A
Explanation

Explanation/Reference:
The function "foo()" is called with 3 arguments:
- A function created by Lambda which multiple the input argument by 2
- A function created by Lambda which divide the input argument by 2 and return the integer part
- The value 2

In the function "foo()", the value 2 is passed to the 2nd Lamba function as argument and returns 2 / 2 = 1
The result returned by the 2nd Lambda function above is passed to the 1st Lambda function as argument and returns 1 * 2 = 2.

Code for testing:

```
def foo(x,y,z):
~~~~return x(y(z))
print( foo(lambda x: 2*x, lambda x: x//2, 2))
```

QUESTION 136
Which one of the platform module functions should be used to determine the Python version?

A. platform.uname()
B. platform.version()
C. platform.python_version()
D. platform.processor()

Correct Answer: C
Explanation

Explanation/Reference:

QUESTION 137
What is the expected behavior of the following code?

```
m = 0

def foo(n):
    global m
    assert m == 0
    try:
        return 1/n
    except ArithmeticError:
        m += 1
        raise

try:
    foo(0)
except ArithmeticError:
    m += 1
except:
    m += 2

print(m)
```

A. it outputs 2
B. the code is erroneous and it will not execute
C. it outputs 1
D. it outputs 3

Correct Answer: A
Explanation

Explanation/Reference:
Similar to Q101. But since the assert condition is "`m == 0`" and is evaluated to be True, no AssertionError is rasied.

Then within the function it attempt to `return 1/n` (where n is the parameter variable accepting the

argument 0 since the function is called with `fn(0)`) and this will raise ZeroDivisionError.

Since ZeroDivisionError is a SubClass of ArithmeticError, it can be handled in the `try` statement inside the function and the global variable `m` is increment by 1. It then the same Exception that has been handled i.e. ZeroDivisionError is raised again.

In the calling statment, the Exception that is rasied again is handled by the `except` clause with ArithmeticError , therefore `m` is incremented by 1 i.e. 1 + 1 = 2.

Code for testing:

```
m = 0

def foo(n):
~~~~global m
~~~~assert m == 0
~~~~try:
~~~~~~~~return 1/n
~~~~except ArithmeticError:
~~~~~~~~m += 1
~~~~~~~~raise

try:
~~~~foo(0)
except ArithmeticError:
~~~~m += 1
except:
~~~~m += 2

print(m)
```

QUESTION 138
Which of the following snippets will execute without raising any unhandled exceptions? (Select two answers)

A.
```
try:
    print(-1/1)
except:
    print(0/1)
else:
    print(1/1)
```

B.
```
try:
    x = 1 / 0
except:
    x = 1 / 1
else:
    x = x + 1
```

C.
```
try:
    x = y / 0
except (NameError, SystemError):
    x = y + 1
else:
    y = x
```

D.
```
try:
    x = 1
except:
    x = x + 1
else:
    x = y + 2
```

Correct Answer: AB
Explanation

Explanation/Reference:
For A, the statement in `try` clause does not raise any exception and the statement in `else` clause does not raise any exception. Therefore there is no unhandled exception.

For B, the statement in `try` clause raise ZeroDivisionError but is handled by `except` clause. The statement in `except` clause does not raise any exception. Therefore there is no unhandled exception.

For C, the statement in `try` clause raise NameError (due to undefined variable "y") but is unhandled by `except` clause.

For D, the statement in `try` clause does not raise any exception . However, the statement in `else` clause raise NameError (due to undefined variable "y") which cannot be handled.

Code for testing:

```
import traceback

print("Choice A")
try:
~~~~print(-1/1)
except:
~~~~print(0/1)
else:
~~~~print(1/1)
print()

print("Choice B")
try:
~~~~x = 1 / 0
except:
~~~~x = 1 / 1
else:
~~~~x = x + 1
print()

print("Choice C")
try:
~~~~try:
~~~~~~~~x = y / 0
~~~~except (NameError, SystemError):
~~~~~~~~x = y + 1   # Unhandled Error!
~~~~else:
~~~~~~~~y = x
except:
~~~~print(traceback.format_exc())
print()

print("Choice D")
try:
~~~~try:
~~~~~~~~x = 1
~~~~except:
~~~~~~~~x = x + 1
~~~~else:
~~~~~~~~x = y + 2   # Unhandled Error!
except:
~~~~print(traceback.format_exc())
print()
```

QUESTION 139
What is the expected behavior of the following code?

```
s='2A'

try:
```

```
    n = int(s)
except:
    n = 3
except ValueError:
    n = 2
except ArithmeticError:
    n = 1

print(n)
```

A. the code is erroneous and it will not execute
B. it outputs 2
C. it outputs 1
D. it outputs 3

Correct Answer: A
Explanation

Explanation/Reference:
A `except` clause without any exception specified must be configured last. Otherwise syntax error will occur and the program will not be run.

Remarks:
If that except clause is removed, the answer is 2 since when `int()` cannot create an `int` object from the string "2A", the exception raised is ValueError.

QUESTION 140
What is the expected behavior of the following code?

```
d = {'1': '1', '2': '2'}

try:
    d['1'] = d['3']
except BaseException as error:
    print(type(error))
```

A. it outputs `<class 'KeyError'>`
B. the code is erroneus and it will not execute
C. it outputs `<class 'Exception'>`
D. it outputs `<class 'BaseException'>`

Correct Answer: A
Explanation

Explanation/Reference:
A KeyError occurs when accessing d['3'] since there is no key '3' in the dictionary "d".
It is handled by the except clause with BaseException. The type of the Exception object is printed and therefore "KeyError" is printed.

QUESTION 141
Which of the following invocations are valid'? (Select two answers)

A. `"python".find("")`
B. `"python".sort()`
C. `sorted("python")`
D. `sort("python")`

Correct Answer: AC

Explanation

Explanation/Reference:
B is wrong since String does not have the `sort()` method.
D is wrong since `sort()` is not a built-in function.

QUESTION 142
Which of the following statements are true? (Select two answers)

A. an escape sequence can be recognized by the # sign put in front of it.
B. UTF-8 is one of the ways of representing UNICODE code points
C. ASCII is the name of a character coding standard
D. a code point is a point inside the code when execution stops immediately

Correct Answer: BC
Explanation

Explanation/Reference:
For A, escape sequence is recognized by backslash character \

QUESTION 143
Assuming that the snippet below has been executed successfully, which of the following expressions will evaluate to True? (Select two answers)

```
string = 'REPTITLE'[:3:]
string = string[-1] + string[-2::1]
```

A. `string` is None
B. `string[0] == 'E'`
C. `len(string) == 3`
D. `string[0] < string[-1]`
E. `string[0] == string[-1]`

Correct Answer: CE
Explanation

Explanation/Reference:
Since "

```
R E P T I T L E
0 1 2 3 4 5 6 7
      |
```

`'REPTITLE'[:3:]` is therefore `'REP'` and assigned to "`string`"

`string[-1]` is "P"
`string[-2::1]` is "EP"
Concatenating them becomes "PEP" and assigned back to "`string`".

Therefore:
C is correct since `len(string)` is 3
E is correct since the first character is the equal to last character since both are "P".

Code for testing:

```
string = 'REPTITLE'[:3:]
print(string)               # REP
string = string[-1] + string[-2::1]
print(string)               # PEP
```

```
print(string is None)           # False
print(string[0] == 'E')         # False
print(len(string) == 3)         # True
print(string[0] < string[-1])   # False
print(string[0] == string[-1])  # True
```

QUESTION 144
What is the expected behavior of the following code?

```
string = str(1/3)
dummy = ''
for character in string:
    dummy = dummy + character
print(dummy[0])
```

A. it outputs 'None'
B. it outputs 3
C. it raises an exception
D. it outputs 0

Correct Answer: D
Explanation

Explanation/Reference:
Similar to Q75 but the first character of "0.3333...." is printed.

Code for testing:

```
string = str(1/3)
dummy = ''
for character in string:
~~~~dummy = dummy + character
print(dummy[0])
```

QUESTION 145
What is the expected behavior of the following code?

```
the_list = "1,2 3".split()
the_string = ''.join(the_list)
print(the_string.isdigit())
```

A. it outputs nothing
B. it raises an exception
C. it outputs True
D. it outputs False

Correct Answer: D
Explanation

Explanation/Reference:
Splitting without any argument i.e. whitespace will return two Substrings "`1,2`" and "3" as elements in the new List.
After joining using empty string as separator, the string becomes "`1,23`"
Since it contains a comma "`,`", `isdigit()` returns `False`

Code for testing:

```
the_list = "1,2 3".split()
print(the_list)             # ['1,2', '3']
```

```
the_string = ''.join(the_list)
print(the_string)           # 1,23
print(the_string.isdigit()) # False
```

QUESTION 146
What is a true about python class constructors? (Select two answers)

A. the constructor cannot be directly invoked under any circumstances.
B. the constructor's first parameter must always be named `self`
C. there can be only one constructor in a Python class
D. the constructor cannot return a result other than None

Correct Answer: CD
Explanation

Explanation/Reference:

QUESTION 147
What is the expected behavior of the following code?

```
class Class:
    Var = 0
    def __init__ (self, var):
        self.var = var
        Class.Var += 1

object_1 = Class(1)
object_2 = Class(2)
print(Class.Var + object_1.var + object_2.var)
```

A. it outputs 2
B. it outputs 3
C. it outputs 5
D. it raises an exception

Correct Answer: C
Explanation

Explanation/Reference:
The class variable "Var" is initialized to 0.

object_1 is created with the argument 1. Since 1 is passed to parameter variable "`var`" in "`__init__`", the instance variable "`var`" of object_1 is set to **1**. Finally the class variable "Var" is incremented by 1 to 1.

object_2 is created with the argument 2. Since 2 is passed to parameter variable "`var`" in "`__init__`", the instance variable "`var`" of object_2 is set to **2**. Finally the class variable "Var" is incremented by 1 to **2**.

By adding the class variable "Var", instance variable "`var`" of object_1 and instance variable "`var`" of object_1, the result is therefore 5.

Code for testing:

```
class Class:
~~~~Var = 0
~~~~def __init__ (self, var):
~~~~~~~~self.var = var
~~~~~~~~Class.Var += 1

object_1 = Class(1)
object_2 = Class(2)
```

```
print(Class.Var + object_1.var + object_2.var)
```

QUESTION 148
What is true about Object-Oriented Programming in Python? (Select two answers)

A. the same class can be used many times to build a number of objects
B. a subclass is usually more specialized than its superclass
C. If a real-life object can be described with a set of adjectives, they may reflect a Python object method
D. each object of the same class can have a different set of methods

Correct Answer: AB
Explanation

Explanation/Reference:
C is wrong since adjectives are used for describing things (i.e. not a behavior / action), they can only stored as states / Instance variables.

D can actually possible but not a simple basic feature and its usage is different from a normal Instance method in some occasions. Since A and B is already correct, D is not chosen as answer.

QUESTION 149
Assuming that the code below has been executed successfully, which of the following expressions evaluate to True? (Select
two answers)

```
class Class:
    data = 1
    def __init__(self, value):
        self.prop = self.var = value

Object = Class(2)
```

A. `'var' in Object.__dict__`
B. `len(Object.__dict__) == 2`
C. `'data' in Object.__dict__`
D. `'var' in Class.__dict__`

Correct Answer: AB
Explanation

Explanation/Reference:
A returns True since "`prop`" and "`var`" are defined as instance variables in "`__init__`" and can therefore be found in `Object.__dict__`
B returns True since there are two insance variables for the object Object.

C returns False since "`data`" is a Class variable and cannot be found in `Object.__dict__`
D returns False since "`var`" is an Instance variable and cannot be found in `Class.__dict__`

Code for testing:

```
class Class:
~~~~data = 1
~~~~def __init__(self, value):
~~~~~~~~self.prop = self.var = value

Object = Class(2)

print('var' in Object.__dict__)     # True
print(len(Object.__dict__) == 2)    # True
print('data' in Object.__dict__)    # False
```

```
print('var' in Class.__dict__)      # False
```

QUESTION 150
Assuming that the following piece of code has been executed successfully, which of the expressions evaluate to True? (Select two answers)

```
class A:
    VarA = 1
    def __init__(self):
        self.prop_a = 1

class B(A):
    VarA = 2
    def __init__(self):
        super().__init__()
        self.prop_b = 2

obj_a = A()
obj_aa = A()
obj_b = B()
obj_bb = obj_b
```

A. `obj_a is obj_aa`
B. `A.VarA == 1`
C. `isinstance(obj_b,A)`
D. `B.VarA == 1`

Correct Answer: BC
Explanation

Explanation/Reference:
A returns False since both `obj_a` and `obj_aa` are created by A() in two different statements and therefore they are two different objects.
B retruns True since the Class variable "`VarA`" in class A is initialized to 1 and is not changed in anywhere else.
C returns True since obj_b is of class B which is a subclass of class A.
D returns False since the Class variable "`VarA`" in class B is initialized to 2 and is not changed in anywhere else.

Code for testing:

```
class A:
~~~~VarA = 1
~~~~def __init__(self):
~~~~~~~~self.prop_a = 1

class B(A):
~~~~VarA = 2
~~~~def __init__(self):
~~~~~~~~super().__init__()
~~~~~~~~self.prop_b = 2

obj_a = A()
obj_aa = A()
obj_b = B()
obj_bb = obj_b

print(obj_a is obj_aa)      # False
print(A.VarA == 1)          # True
print(isinstance(obj_b,A))  # True
print(B.VarA == 1)          # False
```

QUESTION 151
What is the expected behavior of the following code?

```
class Class:
    __Var = 0
    def __foo(self):
        Class._Class__Var += 1
        self.__prop = Class._Class__Var

o1 = Class()
o1.foo()
o2 = Class()
o2.foo()
print(o2._Class__Var + o1._Class__prop)
```

A. It raises an exception
B. it outputs 1
C. it outputs 6
D. it outputs 3

Correct Answer: A
Explanation

Explanation/Reference:
Since the instance method "__foo" is using name mangling, the name `__foo()` becomes `_Class__foo()`
Therefore the 2nd statement and 4th statement in the main program code will cause error.

Remarks:
This program code is special since it entered some mangled names manually.

If the method name is changed from "__foo" to "foo", the result will be 3. Since:
o1 calls `foo()` and this will increments the class variable "__Var" by 1 to 1 and then this value is assigned to Instance variable "__prop" of o1. Therefore "`__prop`" of o1 has a value of **1**.
o2 calls `foo()` and this will increments the class variable "__Var" by 1 to **2** and then this value is assigned to Instance variable "__prop" of o2. Therefore "`__prop`" of o1 has a value of 2.

Adding class variable "_Class__Var" and the Instance variable "_Class`__prop`" of o1 therefore gives 3.

Code for testing (if the method name `__foo` has changed to `foo`)

```
class Class:
~~~~__Var = 0
~~~~def foo(self):
~~~~~~~~Class._Class__Var += 1
~~~~~~~~self.__prop = Class._Class__Var

o1 = Class()
o1.foo()
o2 = Class()
o2.foo()
print(o2._Class__Var + o1._Class__prop)
```

QUESTION 152
Assuming that the code below has been placed inside a file named `code.py` and executed successfully, which of the following expressions evaluate to True? (Select two answers)

```
class ClassA:
    var = 1
    def __init__(self, prop):
        propl = prop2 = prop
```

```
class ClassB(ClassA):
    def __init__(self, prop):
        prop3 = prop ** 2
        super().__init__(prop)
    def __str__(self):
        return 'Object'


Object = ClassA(2)
```

A. `str(Object) == 'Object'`
B. `__name__ == '__main__'`
C. `ClassA.___module__ == '__main__'`
D. `len(ClassB.__bases__) == 2`

Correct Answer: BC
Explanation

Explanation/Reference:
Similar to Q24 except that the __str__ method is defined in ClassB and "Object" is a ClassA object.

A is wrong since ClassA's has no __str__() method and the result of "str()" will be the one inherited from "`object`" which will show "<__main__.ClassA object at ....>"
B is correct since the file is run directly.
C is corrent since ClassA is located in a source file wihch is running directly as the main program code.
D is wrong since ClassB has only one direct SuperClass.

Code for testing:

```
class ClassA:
~~~~var = 1
~~~~def __init__(self, prop):
~~~~~~~~propl = prop2 = prop

class ClassB(ClassA):
~~~~def __init__(self, prop):
~~~~~~~~prop3 = prop ** 2
~~~~~~~~super().__init__(prop)
~~~~def __str__(self):
~~~~~~~~return 'Object'

Object = ClassA(2)

print(str(Object) == 'Object')          # False
print(__name__ == '__main__')           # True
print(ClassA.__module__ == '__main__')  # True
print(len(ClassB.__bases__) == 2)       # False
```

QUESTION 153
What is the expected output of the following code?

```
mytu = ('a', 'b', 'c')
m = tuple(map(lambda x: chr(ord(x) + 1), mytu))
print(m[-2])
```

A. an exception is raised
B. a
C. b
D. c

Correct Answer: D

Explanation

Explanation/Reference:
A Tuple object "`mytu`" is formed using Tuple literal having elements 'a', 'b', 'c'
`map()` uses a Lambda function which tranlate each character by adding 1 to its ASCII code and returns the result as character. (i.e. 'a' will be mapped to 'b' ... etc)
A new Tuple is formed from the `map` object having elements 'b', 'c', 'd' and assigned to `m`
Hence the last 2nd elements of m is therefore 'c'

Code for testing:

```
mytu= ('a', 'b', 'c')
m = tuple(map(lambda x: chr(ord(x) + 1), mytu))
print(m[-2])
```

QUESTION 154
What is the expected output of the following code if there is no file named non-existing_file inside the working directory?

```
try:
    f = open('non_existing_file', 'w')
    print(1, end=' ')
    s = f.readline()
    print(2, end=' ')
except IOError as error:
    print(3 , end=' ')
else:
    f.close()
    print(4, end=' ')
```

A. 1 2 3 4
B. 2 4
C. 1 3
D. 1 2 4

Correct Answer: C
Explanation

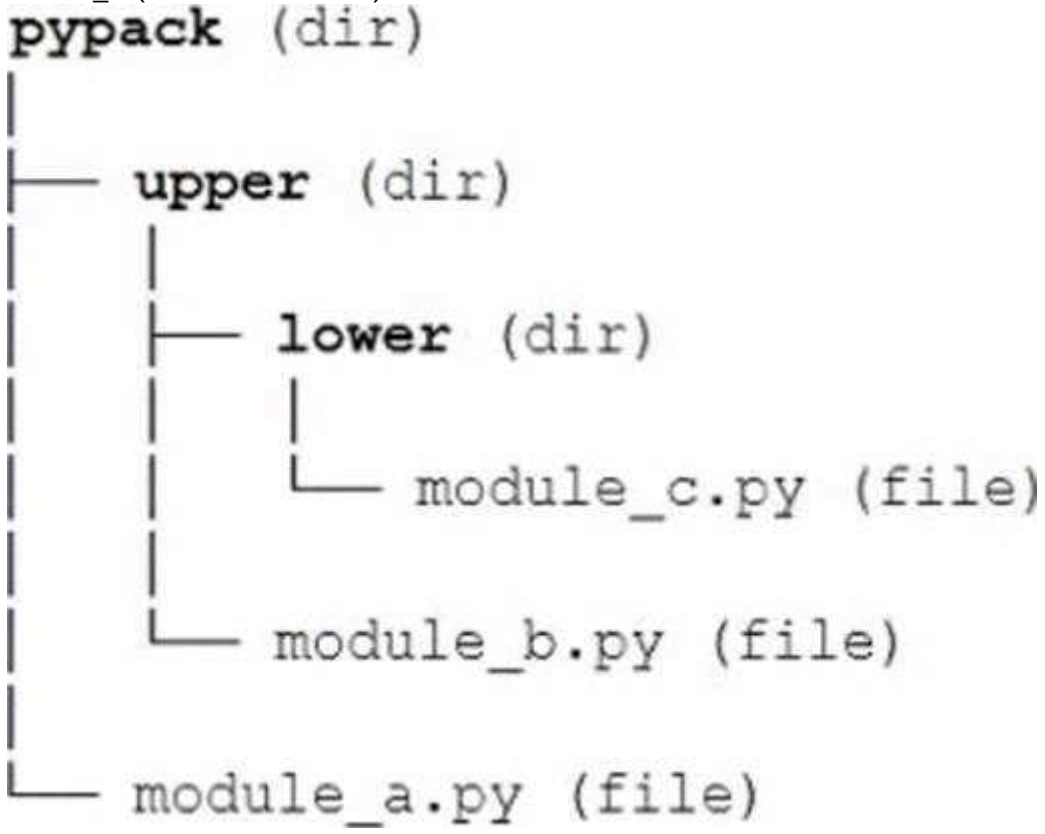
Explanation/Reference:
The program flow is:
- The file specified in `open()` with mode "w" can be existing or non-existing. Therefore no exception occurs and a new file will be created if non-existing.
- "**1**" is printed.
- Exception occurs when you try to read the file opened with mode "w".
- The above is handled by `except` clause with "IOError". Therefore "**3**" is printed.
- Since exception occurs, `else` clause is not run.

Code for testing:

```
try:
~~~~f = open('non_existing_file', 'w')
~~~~print(1, end=' ')
~~~~s = f.readline()
~~~~print(2, end=' ')
except IOError as error:
~~~~print(3 , end=' ')
else:
~~~~f.close()
~~~~print(4, end=' ')
```

QUESTION 155

With regards to the directory structure below, select the proper forms of the directives in order to import module_c. (Select two answers)



A. `import upper.lower.module_c`
B. `import pypack.upper.lower.module_c`
C. `import upper.module_c`
D. `from pypack.upper.lower import module_c`

Correct Answer: BD
Explanation

Explanation/Reference:

QUESTION 156
Assummg that the math module has been successtully imported, which of the following expressions evaluate to True? (Select
answers)

A. `math.hypot(3,4) == math.sqrt(25)`
B. `math.floor(2.5) == math.trunc(2.5)`
C. `math.ceil(2.5) < math.floor(2.5)`
D. `math.ceil(2.5) == math.trunc(2.5)`

Correct Answer: AB
Explanation

Explanation/Reference:
For A, left side is 5 and right side is 5
For B, left side is 2 and right side is 2
For C, left side is 3 and right side is 2
For D, left side is 3 and right side is 2

Code for testing:

```
import math

print(math.hypot(3,4) == math.sqrt(25))    # True
print(math.floor(2.5) == math.trunc(2.5))  # True
print(math.ceil(2.5) < math.floor(2.5))    # False
print(math.ceil(2.5) == math.trunc(2.5))   # False
```

QUESTION 157
What is the expected output of the following code?

```
import sys
import math

b1 = type(dir(math)) is list
b2 = type(sys.path) is list
print(b1 and b2)
```

A. 0
B. False
C. None
D. True

Correct Answer: D
Explanation

Explanation/Reference:
`dir(sys)` returns a List object and therefore b1 is True
`sys.path` returns a List object and therefore b2 is True

You can copy the above code for testing.

QUESTION 158
What is the expected behavior of the following code?

```
my_tuple = (1, 2, 3)

try:
   my_tuple[3] = my_tuple[2]
except IndexError as error:
   x = error
except Exception as exception:
   x = exception
else:
   x = None
print(x)
```

A. it outputs 'tuple' object does not support item assignment
B. the code is erroneous and it wil! not execute
C. it outputs None
D. it outputs list assignment index out of range

Correct Answer: A
Explanation

Explanation/Reference:
Although the statement in the `try` clause seems to raise an IndexError, however before evaluating the Index number, another exception is raised. It is the TypeError about "'tuple' object does not support item assignment" since it involves assignment to a immutable Tuple object

The TypeError is handled by the second except clause which stores the Exception object to x.

By printing x, the message of the Exception object stored in x will be shown.

Code for testing:

```
my_tuple = (1, 2, 3)

try:
~~~~my_tuple[3] = my_tuple[2]
except IndexError as error:
~~~~x = error
except Exception as exception:
~~~~x = exception
else:
~~~~x = None
print(x)           # 'tuple' object does not support item assignment
print()
print()
print(type(x))     # <class 'TypeError'>  (additional info proving it is
TypeError)
```

QUESTION 159
What is the expected behavior of the following code?

```
s='2A'

try:
    n = int(s)
except ValueError:
    n = 2
except:
    n = 3
except ArithmeticError:
    n = 1

print(n)
```

A. the code is erroneous and it will not execute
B. it outputs 2
C. it outputs 1
D. it outputs 3

Correct Answer: A
Explanation

Explanation/Reference:
Similar to Q139. The except without any exception specified must be the last `except` clause.

QUESTION 160
Which of the following statements are true? (Select twg answers)

A. an escape Sequence can be recognized by the / sign put in front of it
B. a code point is a number assigned to a given character
C. ASCII is a subset of UNICODE
D. II in ASCII stands for Internal Information

Correct Answer: BC
Explanation

Explanation/Reference:
A is wrong since escape Sequence uses backslash \ instead of a forwardslash /
D is wrong since ASCII stands for American Standard Code for **Information Interchange**

QUESTION 161
Which of the following invocations are valid'? (Select two answers)

A. `"python".index("th")`
B. `"python".sort()`
C. `sorted("python")`
D. `rfind("python","r")`

Correct Answer: AC
Explanation

Explanation/Reference:
B is wrong since String does not have `sort()` method.
D is wrong since rfind() is not a built-in function, it must be called by using a String object as qualifier

QUESTION 162
Which of the following expressions evaluate to True? (Select two answers)

A. `121 + 1 != '1' + 2 + '2'`
B. `'3.14' != str(3.1415)`
C. `'1' + '1' + '1' < '1' + '3'`
D. `'AbC'.lower() < 'AB'`

Correct Answer: BC
Explanation

Explanation/Reference:
A is wrong since, on the right side of "=", string and int value cannot be concatenate / add with "+".
B is correct since the string `'3.14'` is different from string `'3.1415'` (obtained from `str(3.1415)`)
C is correct since `'111'` is smaller than `'13'` since the 2nd character of the former string i.e. '1' is smaller than the 2nd character of the latter string i.e. '3'. (Note that length only matters if all characters in the shorter string are the same to those in the longer string)
D is wrong since the string `'AbC'.lower()` i.e. `'abc'` is larger than the string `'AB'` since lowercase `'a'` of the 1st character in the former string is acutally larger than uppercase `'A'` of the 1st character of the later string.

Code for testing:

```
# print(121 + 1 != '1' + 2 + '2')   # Error
print('3.14' != str(3.1415))        # True
print('1' + '1' + '1' < '1' + '3')  # True
print('AbC'.lower() < 'AB')         # False
```

QUESTION 163
What is a true about python class constructors? (Select two answers)

A. the constructor's first parameter identifies an object currently being created
B. super-class constructor is invoked implicitly during constructor execution
C. the constructor can be invoked directly under strictly defined circumstances
D. the constructor cannot use the default values of the parameters

Correct Answer: AC
Explanation

Explanation/Reference:
For B, Python requires you to call `super().__init__()` explicitly in order to call Super-Class constructor.

For C, `__init__()` is normally called implicity by e.g. obj = ClassA(). You can use an existing object to call `obj.__init__()`, the value of the instance variables will be reset to initial value. Therefore it is invoked directly only under special circumstances..

For D, the parameter in `__init__()` can be setup with default values.

QUESTION 164
What is the expected output of the following snippet?

```
class Upper:
    def __init__(self):
        self.property = 'upper'

class Lower(Upper):
    def __init__(self):
        super().__init__()

Object = Lower()
print(isinstance(Object,Lower), end=' ')
print(Object.property)
```

A. False upper
B. False lower
C. True upper
D. True lower

Correct Answer: C
Explanation

Explanation/Reference:
When creating the Lower object, the`__init__` in the class Lower uses "`super().__init__()`" to call the `__init__` in the parent class "Upper" which define the Instance variable "`property`" with the intial value "upper".

Therefore:
The object is of class Lower, isintance() will return **True**.
The Instance variable "property" is assigned with "**upper**" as mentioned above.

Code for testing:

```
class Upper:
~~~~def __init__(self):
~~~~~~~~self.property = 'upper'

class Lower(Upper):
~~~~def __init__(self):
~~~~~~~~super().__init__()

Object = Lower()
print(isinstance(Object,Lower), end=' ')
print(Object.property)
```

QUESTION 165
Assuming that the following inheritance set is in force, which of the following classes are declared properly? (Select two answers)

```
class A:
    pass

class B(A):
    pass

class C(A):
    pass
```

A. `class class_3(A,C): pass`
B. `class class_2(B,C): pass`
C. `class class_1(C,B): pass`
D. `class class_4(A,B): pass`

Correct Answer: BC
Explanation

Explanation/Reference:
A and D is wrong is you cannot put a parent class before a child class in the Superclass list.

QUESTION 166
Assuming that the following code has been executed successfully, select the expressions which evaluate to True (Select two answers)

```
def f(x):
    def g(x):
        return x * x
    return g

a = f(2)
b = f(3)
```

A. `a is not None`
B. `a(2) == 4`
C. `a == b`
D. `b(1) == 4`

Correct Answer: AB
Explanation

Explanation/Reference:
The function `f()` returns the nested function object.

However the nested function g() will not get the value of "x" in `f()` with `g()` has defined the parameter variable (a local variable) "x" having the same name again.

Therefore the argument 2 and 3 passed to `f()` for obtaining the nested object is irrelevant.
Note that 2 and 3 passed to `f()` will returns two different nested function objects (even the value 2 and 3 passed are not actually used by g()).

C is wrong since a and b are two different nested function objects.
D is wrong since b(1) will return 1 * 1 = 1.

Code for testing:

```
def f(x):
~~~~def g(x):
~~~~~~~~return x * x
~~~~return g

a = f(2)
```

```
b = f(3)

print(a(5))   # prints 25
print(b(5))   # prints 25

print(a is not None)    # True
print(a(2) == 4)        # True
print(a == b)           # False
print(b(1) == 4)        # False
```

QUESTION 167
What is the expected output of the following code?

```
def foo(x,y):
  return y(x) + y(x+1)

print( foo(1, lambda x: x*x) )
```

A. 5
B. 3
C. 4
D. an exception is raised

Correct Answer: A
Explanation

Explanation/Reference:
The function "foo()" is called with 2 arguments:
- The value 1
- A function created by Lambda which multiply the input argument by itself i.e. the square of the input arugment

For the function "foo()", it returns the adding of two values:
- The 1st value is the the argument 1 is passed to the Lamba function as argument and returns 1 * 1 = **1**
- The 2nd value is the argument 1 added by 1 i.e. 1 + 1 = 2 and then passed to the Lamba function as argument and returns 2 * 2 = **4**
The sum returned is therefore 1 + 4 = **5**

Code for testing:

```
def foo(x,y):
~~~~return y(x) + y(x+1)
print( foo(1, lambda x: x*x) )
```

QUESTION 168
What is the expected output of the following code if there is no file named non-existing_file inside the working directory?

```
try:
    f = open('non_existing_file', 'r')
    print(1, end=' ')
except IOError as error:
    print(error.errno, end=' ')
    print(2, end=' ')
else:
    f.close()
    printt(3, end=' ')
```

A. 2 2 3
B. 1 2 2

C. 1 3
D. 2 2

Correct Answer: D
Explanation

Explanation/Reference:
The program flow is:
- Exception occurs when opening a non existing file with mode "r".
- Exception is handled by the `except` clause with IOError.
- The error number is printed. (The error number for non-existing file is 2)
- The value "2" is printed.
- `else` clause not run.

Hence the answer should contain two numbers with "2" as the last number.

Code for testing:
(You must ensure that the file `'non_existing_file2'` does not exist)

```
try:
~~~~f = open('non_existing_file2', 'r')
~~~~print(1, end=' ')
except IOError as error:
~~~~err = error
~~~~print(error.errno, end=' ')    # 2
~~~~print(2, end=' ')              # 2
else:
~~~~f.close()
~~~~print(3, end=' ')

print()
print(error)   # Errno 2] No such file or directory: 'non_existing_file2'
```

Extra Reference:
The scoping of the variable used for storing exception object i.e. "`error`" in the `except` clause is very special.
This variable "error" will disappear / undefined automatically after the `except` clause.
Hence you cannot access this variable for the Exception object outside of the except clause unless you assign it to another variable e.g. "`err`" shown above

QUESTION 169
What is the expected out of the following code of the file named non_zero_length_existing_text_file is a non-zero length file located inside the working directory?

```
try:
    f = open('non_zero_length_existing_text_file', 'rt')
    d = f.read(1)
    print(len(d))
    f.close()
except IOError:
    print(-1)
```

A. an errno value corresponding to file not found
B. -1
C. 1
D. 0

Correct Answer: C
Explanation

Explanation/Reference:
Since it is an existing file, no exception occurs when opening with mode "rt".
Scine a byte / character is read, the length of the string "d" being printed is "**1**".

QUESTION 170
What is the expected output of the following code?

```
myli = [1, 2, 4]
m = list(map(lambda x: 2**x, myli))
print(m[-1])
```

A. an exception is raised
B. 1
C. 4
D. 16

Correct Answer: D
Explanation

Explanation/Reference:
A List object "`myli`" is formed using List literal having elements 1, 2, 4
`map()` uses a Lambda function which tranlate each value by squaring it.

A new List is formed from the `map` object having elements 1, 4, 16 and assigned to `m`
Hence the last elements of m is therefore 16

Code for testing:

```
myli = [1, 2, 4]
m = list(map(lambda x: 2**x, myli))
print(m[-1])
```

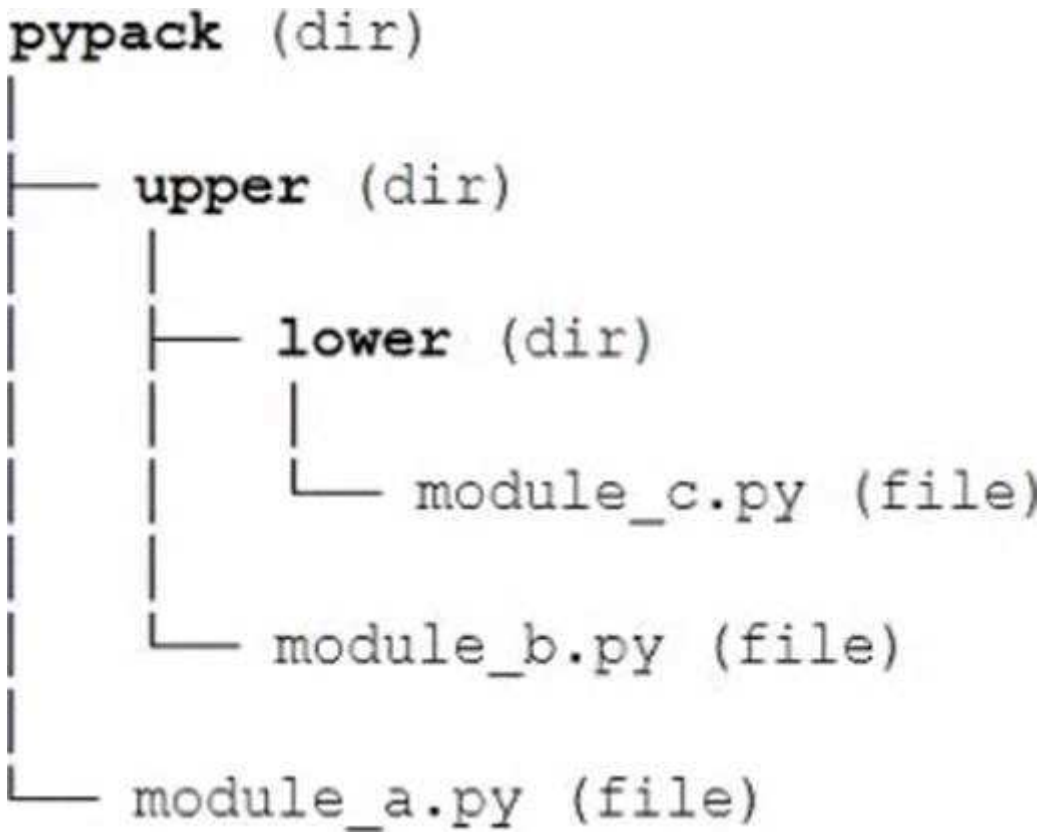
QUESTION 171
What is true about lambda functions? (Select two answers)

A. they cannot return the None value as a result
B. they must not contain the return keyword
C. they are called anonymous functions
D. they must have a non-zero number of parameters

Correct Answer: BC
Explanation

Explanation/Reference:

QUESTION 172
With regards to the directory structure below, select the proper forms of the directives in order to import module_b. (Select two answers)



A. `import pypack.upper.module_b`
B. `from pypack.upper import module_b`
C. `import module_b`
D. `from upper.module_b`

Correct Answer: AB
Explanation

Explanation/Reference:

QUESTION 173
Which of the following snippets will execute without raising any unhandled exceptions? (Select two answers)

A.
```
try:
    print(1/0)
except ValueError:
    print(1/1)
else:
    print(1/2)
```
B.
```
try:
    print(0/1)
except:
    print(1/1)
else:
    print(2/1)
```
C.
```
try:
    print(float("1e1"))
except (ValueError, NameError):
    print(float("1a1"))
else:
    print("101")
```
D.
```
try:
```

```
        print(1/1)
    except:
        print(2/1)
    else:
        print(3/0)
```

Correct Answer: BC
Explanation

Explanation/Reference:
For A, the statement in `try` clause raise ZeroDivisionError can cannot be unhandled by `except` clause.

For B, the statement in `try` clause does not raise any exception and the statement in `else` clause does not raise any exception. Therefore there is no unhandled exception.

For C, the statement in `try` clause does not raise any exception and the statement in `else` clause does not raise any exception. Therefore there is no unhandled exception.

For D, the statement in `try` clause does not raise any exception. But the statement in `else` clause raise the ZeroDivisionError and cannot be handled.

Code for testing:

```
import traceback

print("Choice A:")
try:
~~~~try:
~~~~~~~~print(1/0)   # Unhandled Error!
~~~~except ValueError:
~~~~~~~~print(1/1)
~~~~else:
~~~~~~~~print(1/2)
except:
~~~~print(traceback.format_exc())
print()

print("Choice B:")
try:
~~~~print(0/1)
except:
~~~~print(1/1)
else:
~~~~print(2/1)
print()

print("Choice C:")
try:
~~~~print(float("1e1"))
except (ValueError, NameError):
~~~~print(float("1a1"))
else:
~~~~print("101")
print()

print("Choice D:")
try:
~~~~try:
~~~~~~~~print(1/1)
~~~~except:
~~~~~~~~print(2/1)
~~~~else:
~~~~~~~~print(3/0)   # Unhandled Error!
except:
~~~~print(traceback.format_exc())
```

```
print()
```

QUESTION 174
Which of the following expression evaluate to True? (Select two answers)

A. `len("'") == 2`
B. `len('''12`
`34''') == 4`
C. `chr(ord'Z') - 1) == 'Y'`
D. `ord("0") - ord("9") == 10`

Correct Answer: AC
Explanation

Explanation/Reference:
A is correct since there are two characters of '
C is correct since the character just before of Z is Y.

B is wrong since there are a total of 5 characters since a newline character is included between "12" and "34".
D is wrong since the digit 0 is before the digit 9 in ASCII code and therefore the result is a negative number.

Code for testing:

```
print(len("'") == 2)               # True
print(len('''12
34''') == 4)                       # False
print(chr(ord('Z') - 1) == 'Y')    # True
print(ord("0") - ord("9") == 10)   # False
```

QUESTION 175
What is the expected behavior of the following code?

```
class Super:
    def make(self):
        return 0
    def doit(self):
        return self.make()

class Sub_A(Super):
    def make(self):
        return 1

class Sub_B(Super):
    def make(self):
        return 2

a = Sub_A()
b = Sub_B()
print(a.doit() + b.doit())
```

A. it raises an exception
B. it outputs 2
C. it outputs 1
D. it outputs 3

Correct Answer: D
Explanation

Explanation/Reference:
a is an object of class Sub_A. When it is used for calling `doit()`, the inherited method defined in the class Super is run and calls `self.make()`. Since Instance method has to be evaluated independently using the current object and there is a Instance method `make()` defined in class Sub_A, this method is run and returns **1**.

b is an object of class Sub_B. When it is used for calling `doit()`, the inerited method defined in the class Super is run and calls `self.make()`. Since Instance method has to be evaluated independently using the current object and there is a Instance method `make()` defined in class Sub_B, this method is run and returns **2**.

The sum of two `doit()` is therefore 1 + 2 = **3**.

Code for testing:

```
class Super:
~~~~def make(self):
~~~~~~~~return 0
~~~~def doit(self):
~~~~~~~~return self.make()

class Sub_A(Super):
~~~~def make(self):
~~~~~~~~return 1

class Sub_B(Super):
~~~~def make(self):
~~~~~~~~return 2

a = Sub_A()
b = Sub_B()
print(a.doit() + b.doit())
```

QUESTION 176
What is true about Object-Oriented Programming in Python? (Select two answers)

A. an object is a recipe for an class
B. encapsulation allows you to protect some data from uncontrolled access
C. the arrows on a class diagram are always directed from a superclass towards its subclass
D. inheritance is the relation between a superclass and a subclass

Correct Answer: BD
Explanation

Explanation/Reference:
B is not straightly correct since there is no implicit access control protection by Python. However since A and C must be wrong, the only choice left is B and D.

QUESTION 177
Which of the following expressions evaluate to True? (Select two answers)

A. `121 + 1 == int('1' + 2 * '2')`
B. `float('3.14') == str('3.' + '14')`
C. `'8' + '8' != 2 * '8'`
D. `'xYz'.lower() > 'XY'`

Correct Answer: AD
Explanation

Explanation/Reference:

A is correct. On the left, 121 + 1 = 122. On the right, "*" is performed first and therefore 2 * '2' becomes '22' and then it is appended to '1' becoming '122' and then it is converted to `int` i.e. 122.
B is wong since the left returns a `float` value but the right returns a `str`.
C is wrong since the left is '8' + '8' = '88' and the right is 2 * '8' i.e. '88'. They are actually equal and therefore "`!=`" returns False.
D is correct since the 1st character on the left is "x" and the 1st character on the right is "X". The uppercase character on the right is smaller than the lowercase character on the left.

Code for testing:

```
print(121 + 1 == int('1' + 2 * '2'))        # True
print(float('3.14') == str('3.' + '14'))    # False
print('8' + '8' != 2 * '8')                 # False
print('xYz'.lower() > 'XY')                 # True
```

QUESTION 178
Which of the following expression evaluate to True? (Select two answers)

A. `len('\'') == 1`
B. `len("""`
`""") > 0`
C. `chr(ord('a') + 1) == 'B'`
D. `ord("z") - ord("Z") == ord("0")`

Correct Answer: AB
Explanation

Explanation/Reference:
A is correct since the string contains only 1 character `'`
B is corrent since the string contains a newline character

C is wrong since the next character of 'a' is 'b' (lowercase)
D is wrong since the result of `ord("z") - ord("Z")` is 32. The code point of the letter "O" is 79. Even the string on the right is a zero digit "0", it is still wrong since the code point of zero "0" is 48.

Remarks : The ASCII character for the code point 32 is a space character.

Code for testing:

```
print(len('\'') == 1)                   # True
print(len("""
""") > 0)                               # True
print(chr(ord('a') + 1) == 'B')         # False
print(ord("z") - ord("Z") == ord("0"))  # False
```

QUESTION 179
What is the expected behavior of the following code?

```
the_string = ",,".join(('alpha', 'omega'))
the_list = the_string.split(',')
print(',' in the_list)
```

A. it outputs nothing
B. it raises an exception
C. it outputs True
D. it outputs False

Correct Answer: D
Explanation

Explanation/Reference:
After a string is splitted by a character, the new list object should not contain that character any more.

The 1st statement uses ",," to join a Tuple `('alpha', 'omega')`. Hence the output is "`alpha,,omega`"
When it is splitted by ",", the List will be `['alpha', '', 'omega']`
Hence, there will not be any element containing ',' in the List.

Code for testing:

```
the_string = ",,".join(('alpha', 'omega'))
print(the_string)     # alpha,,omega
the_list = the_string.split(',')
print(the_list)       # ['alpha', '', 'omega']
print(',' in the_list)  # False

# additional
print()
print()
the_string2 = ",,,".join(('alpha', 'omega'))
print(the_string2)     # alpha,,,omega
the_list2 = the_string2.split(',')
print(the_list2)       # ['alpha', '', '', 'omega']
print(',' in the_list2)  # False
```

QUESTION 180
Assuming that the code below has been executed successfully, which of the following expressions evaluate to True? (Select
two answers)

```
class Class:
    data = 1
    def __init__(self, value):
        self.prop = self.var = value

Object = Class(2)
```

A. `'var' in Object.__dict__`
B. `len(Object.__dict__) == 2`
C. `'data' in Object.__dict__`
D. `'var' in Class.__dict__`

Correct Answer: AB
Explanation

Explanation/Reference:
A returns True since "`Object`" created from the class Class has been defined with the Instance variables "`prop`" and "`var`" inside `__init__`
B returns True since "`Object`" created from the class Class has been defined with two Instance variables i.e. "`prop`" and "`var`" inside `__init__`
C returns False since "`data`" is defined as Class variable and will not be shown in `__dict__` of an object.
D returns False since "`var`" is defined as Instance variable and will not be shown in `__dict__` of a class..

Code for testing:

```
class Class:
~~~~data = 1
~~~~def __init__(self, value):
~~~~~~~~self.prop = self.var = value

Object = Class(2)

print('var' in Object.__dict__)   # True
```

```
print(len(Object.__dict__) == 2)  # True
print('data' in Object.__dict__)  # False
print('var' in Class.__dict__)    # False
```

QUESTION 181
What is the expected behavior of the following code?

```
class Class:
    __Var = 0
    def foo(self):
        Class._Class__Var += 1
        self.__prop = Class._Class__Var

o1 = Class()
o1.foo()
o2 = Class()
o2.foo()
print(o2._Class__Var + o1._Class__prop)
```

A. It raises an exception
B. it outputs 1
C. it outputs 6
D. it outputs 3

Correct Answer: D
Explanation

Explanation/Reference:
The Class variable `__Var` is defined and is changed to `_Class__Var` automatically due to Name Mangling. Its value is initialized to 0

When the object `o1` calls `foo()`, the above class variable is incremented by 1 to 1. Then the value of the class variable i.e. 1 is used as the inital value of the Instance variable "`__prop`" i.e. "`_Class__prop`" of the object `o1`.

When the object `o2` calls `foo()`, the above class variable is incremented by 1 to 2. Then the value of the class variable i.e. 2 is used as the inital value of the Instance variable "`__prop`" i.e. "`_Class__prop`" of the object `o2`.

`o2._Class__Var` gets the value of the Class variable which is **2** and `o1._Class__prop` gets the value of the Instance variable "`prop`" of `o1` which is **1**. Hence the sum is **2** + **1** = **3**.

Code for testing:

```
class Class:
~~~~__Var = 0
~~~~def foo(self):
~~~~~~~~Class._Class__Var += 1
~~~~~~~~self.__prop = Class._Class__Var

o1 = Class()
o1.foo()
o2 = Class()
o2.foo()
print(o2._Class__Var + o1._Class__prop)
```

QUESTION 182
What is the expected behavior of the following code?

```
my_list = [i for i in range(5)]
m = [my_list[i] for i in range(5) if my_list[i] % 2 != 0]
print(m)
```

A. the code is erroneus and it will not execute
B. it outputs [0, 2, 4]
C. it outputs [1, 3]
D. it outputs [0, 1, 2, 3, 4]

Correct Answer: C
Explanation

Explanation/Reference:
In the 1st statement, `[i for i in range(5)]` produces the List object `[0, 1, 2, 3, 4]` for assigning to `my_list`

In the 2nd satement, the same sequence is generated again but each of the number is used as a index for obtaining the elements in my_list. However, my_list has elements which happen to be the same as their Index numbers. When finding the elements not divisible by 2, both index 1 having element 1 and Index 3 having element 3 meet the condition. These two elements are used to form a new List "m".

Hence, m is `[1, 3]`

Code for testing:

```
my_list = [i for i in range(5)]
m = [my_list[i] for i in range(5) if my_list[i] % 2 != 0]
print(m)         # [1, 3]
```

QUESTION 183
Which of the following statements are true? (Select two answers)

A. if `open()`'s second argument is `"r"`, the file must exist or open will fail
B. the second `open()` argument describes the open mode and defaults to `"w"`
C. closing an opened file is performed by the closefile() function
D. if `open()`'s second argument is `"w"` and the invocation succeeds, the previous file's content is lost

Correct Answer: AD
Explanation

Explanation/Reference:
B is wrong since default is "`r`" (or more specifically "`rt`")
C is wrong since the method for closing a file is "`close()`".

QUESTION 184
What is the expected behavior of the following code?

```
def foo(x):
    return -x if x > 0 else x
print(foo(-2))
```

A. it outputs 0.0
B. it outputs 2.0
C. it outputs -2
D. the code is erroneus and it will not execute

Correct Answer: C
Explanation

Explanation/Reference:
In the function, the returning value depends on an `if` expression.

Since -2 is passed as srgument to the local parameter variable `x`, `x > 0` will returns False. Hence instead of returning `-x`, the `x` after `else` will be returned. Since `x` is -2, -2 is returned.

Code for testing:

```
def foo(x):
~~~~return -x if x > 0 else x
print(foo(-2))     # -2
```

QUESTION 185
Assuming that the code below has been executed successfully, which of the following expressions evaluate to True? (Select two answers)

```
class Class:
    var = 1
    def __init__(self, value):
        self.prop = value

Object = Class(2)
```

A. `'prop' in Class.__dict__`
B. `len(Object.__dict__) == 1`
C. `'var' in Object.__dict__`
D. `'var' in Class.__dict__`

Correct Answer: BD
Explanation

Explanation/Reference:
A returns `False` since "`prop`" is defined within the constructor using self as qualifier and therefore it is an Instance variable of the object instead of the Class.
B returns `True` since there is only one Instance variable defined for the object.
C returns False since "`var`" is Class variable (i.e. it is not an Instance variable)
D returns `True` since "var" is a Class variable.

Code for testing:

```
class Class:
~~~~var = data = 1
~~~~def __init__(self, value):
~~~~~~~~self.prop = value

Object = Class(2)

print('prop' in Class.__dict__)    # False
print(len(Object.__dict__) == 1)   # True
print('var' in Object.__dict__)    # False
print('var' in Class.__dict__)     # True
```

QUESTION 186
Which of the following lines of code will work flawlessly when put independently inside the `add_new ( )` method in order to make the
snippet's output equal to `[0, 1, 2]`? (Select two answers)

```
class MyClass:
    def __init__(self, size):
        self.queue = [i for i in range(size)]

    def get(self):
        return self.queue

    def get_last(self):
        return self.queue[-1]

    def add_new(self):
        # insert the line of code here

Object = MyClass(2)
Object.add_new()
print(Object.get())
```

A. `queue.append(self.get_last() + 1)`
B. `self.queue.append(self.queue[-1] + 1)`
C. `self.queue.append(get_last() + 1)`
D. `self.queue.append(self.get_last() + 1)`

Correct Answer: BD
Explanation

Explanation/Reference:
When the object "Object" is being created, a value "2" is passed to paramter "`size`" of the contructor. `[i for i in range(size)]` will therefore create a List `[0, 1]` and assign it to the Instance variable "queue".

After calling the Instance method "`add_new()`", the instance method "`get()`" should returned the List "`[0, 1, 2]`" for printing as required by the question.

Hence, the method "`add_new()`" needs to add the element "2" to the List stored by "`queue`". This can be achieved by one of the followings:
- Get the last element in "`queue`" i.e. 1 through "`self.queue[-1]`", add 1 to it to become 2. Then append the result to the List object "`self.queue`" through "`append()`".
- You can call "`self.get_last()`" which will also return "`self.queue[-1]`" i.e. 1. Then as before, add 1 to it to become 2. for appending to the List object "`self.queue`" through "`append()`".

Note that the other two choices are wrong since the object being used as qualifier for accessing the Instance variable "queue" or the Instance method "`get_last()`" is missing.

Code for testing:

```
class MyClass:
~~~~def __init__(self, size):
~~~~~~~~self.queue = [i for i in range(size)]

~~~~def get(self):
~~~~~~~~return self.queue

~~~~def get_last(self):
~~~~~~~~return self.queue[-1]

~~~~def add_new(self):
~~~~~~~~# self.queue.append(self.get_last() + 1)
~~~~~~~~# self.queue.append(self.queue[-1] + 1)

Object = MyClass(2)
Object.add_new()
print(Object.get())
```

Then you can remove the comment in just one of the lines within `add_new()` for testing.

QUESTION 187
What is the expected output of the following code?

```
import sys
import math

b1 = type(dir(math)[0]) is str
b2 = type(sys.path[-1]) is str
print(b1 and b2)
```

A. 0
B. False
C. None
D. True

Correct Answer: D
Explanation

Explanation/Reference:
Similar to Q157 except that an index number is appended after the result obtained from both `dir()` and `sys.path`. This retrieves an element from each of the two List objects and both elements are String.

Since b1 is True and b2 is True, the result after "`and`" operation is also True.

You can copy the above code for testing.

QUESTION 188
A Python package named `pypack` includes a module named `pymod.py` which contains a function named `pyfun()`.

Which of the following snippets will let you invoke the function? (Select two answers)

A.
```
import pypack
import pypack.pymod
pypack.pymod.pyfun()
```
B.
```
from pypack import *
pyfun()
```
C.
```
from pypack.pymod import pyfun
pyfun()
```
D.
```
import pypack
pymod.pyfun()
```

Correct Answer: AC
Explanation

Explanation/Reference:
A can import the module "pypack.pymod" and you can then use the imported name as qualifier to access the function. Note that the first `import` statement is not relevant and can be removed.

C can import the function `pyfun()` by specifying "`pypack.pymod`" as from. Since the function is imported directly, you can call it without any qualifier.

B is wrong since the import statement can only import module and can only be successfully if "`__all__`" is defined in "`__init__.py`" (e.g. "`__all__ = ["pymod"]`") . Even the module is loaded this way, you still need to specify "`pymod`" as the qualifier in order to call "`pyfun()`".

D is wrong since the import statement can only import module if "`import`" statement is defined in

"__init__.py" (e.g. "import pypack.pymod" or "from. import pymod"). Even the module is loaded this way, you still need to specify "pypack.pymode" as the qualifier in order to call "pyfun()".

QUESTION 189
What is true about the following snippet? (Select two answers)

```
class E(Exception):
    def __init__(self, message):
        self.message = message
    def __str__(self):
        return "it's nice to see you"

try:
    print("I feel fine")
    raise E("what a pity")
except E as e:
    print(e)
else:
    print("the show must go on")
```

A. the code will raise an unhandled exception
B. the string `I feel fine` will be seen
C. the string `it's nice to see you` will be seen
D. the string `what a pity` will be seen

Correct Answer: BC
Explanation

Explanation/Reference:
This is the modified question mentioned in Q70.

Since the "raise" statement is changed from Q70 to trigger the custom exception, B and C are the answer.
Since the __str__ in the custom exception overrides the parent's implementation. The returned string "it's nice to see you" is printed by print(e) in the except suite.

Sample code for this can be found in Q70's Remarks.

QUESTION 190
Which of the following snippets will execute without raising any unhandled exceptions? (Select two

answers)

A.
```
try:
    print(-1/1)
except:
    print(0/1)
else:
    print(1/1)
```
B.
```
try:
    x = 1 / 0
except NameError:
    x = 1 / 1
else:
    x = x + 1
```
C.
```
try:
    x = y + 1
except (NameError, SystemError):
    x = y + 1
else:
    y = x
```
D.
```
try:
    x = 1
except:
    x = x + 1
else:
    x = x + 2
```

Correct Answer: AD
Explanation

Explanation/Reference:
For A, the statement in `try` clause and `else` clause does not raise any exception and the statement in `else` clause does not raise any exception. Therefore there is no unhandled exception.

For B, the statement in `try` clause raise ZeroDivisionError but is NOT handled by `except` clause which specify NameError only..

For C, the statement in `try` clause raise NameError (due to undefined variable "y") but is handled by `except` clause. However, an NameError due to undefined variable "y" occurs again in the `except` clause.

For D, the statement in `try` clause and `else` clause does not raise any exception . Therefore there is no unhandled exception

Code for testing:

```
import traceback

print("Choice A")
try:
~~~~print(-1/1)
except:
~~~~print(0/1)
else:
~~~~print(1/1)
print()

print("Choice D")
try:
~~~~x = 1
except:
~~~~x = x + 1
else:
~~~~x = x + 2
print()
```

```
print("Choice B")
try:
~~~~try:
~~~~~~~~x = 1 / 0                    # Unhandled Error!
~~~~except NameError:
~~~~~~~~x = 1 / 1
~~~~else:
~~~~~~~~x = x + 1
except:
~~~~print(traceback.format_exc())
print()

print("Choice C")
try:
~~~~try:
~~~~~~~~x = y + 1
~~~~except (NameError, SystemError):
~~~~~~~~x = y + 1                    # Unhandled Error!
~~~~else:
~~~~~~~~y = x
except:
~~~~print(traceback.format_exc())
print()
```

QUESTION 191
What is the expected behavior of the following code?

```
the_list = "alpha;beta:gamma".split(":")
the_string = ''.join(the_list)
print(the_string.isalpha())
```

A. it outputs nothing
B. it raises an exception
C. it outputs True
D. it outputs False

Correct Answer: C
Explanation

Explanation/Reference:
Similar to Q117. However all `":"` (colon) in that question are now replaced with `";"` (semicolon)

When splitting with `";"`, since the words are separated by semicolon `";"`, only 3 Substrings is formed and the List contains three elements "`alpha`", "`beta`" and "`gama`"
After joining, the result string will be "`alphabetagamma`".
Since all characters in the joined String are alphabets, `isalpha()` which checks for alphabet returns `True`

Code for testing:

```
the_list = "alpha;beta;gamma".split(";")
print(the_list)                 # ['alpha', 'beta', 'gamma']
the_string = ''.join(the_list)
print(the_string)               # alphabetagamma
print(the_string.isalpha())     # True
```

QUESTION 192
Which of the following expression evaluate to True? (Select two answers)

A. `'not' not in 'in'`
B. `'a' not in 'ABC'.lower()`
C. `'t'.upper() in 'Thames'`

D. `'in not' in 'not'`

Correct Answer: AC
Explanation

Explanation/Reference:
A is correct since `'not'` cannot be found in `'in'`
c is correct since `'t'` after changing to `'T'` can be found within `'Thames'`

Code for testing:

```
print('not' not in 'in')        # True
print('a' not in 'ABC'.lower()) # False
print('t'.upper() in 'Thames')  # True
print('in not' in 'not')        # False
```

QUESTION 193
The __bases__ property contains?

A. base dass ids (`int`)
B. base class locations (`addr`)
C. base class names (`str`)
D. base class objects (`class`)

Correct Answer: D
Explanation

Explanation/Reference:
"`__bases__`" returns the classes' objects (in form of type objects) in a Tuple

The following shows an example of "`print(RacingCar.__bases__)`":
`(<class '__main__.Car'>,)`

QUESTION 194
What is the expected behavior of the following code?

```
class Class:
    Var = 0
    def __foo(self):
        Class.Var += 1
        return Class.Var

o = Class()
o.__Class_foo()
print(o.__Class_foo())
```

A. It raises an exception
B. it outputs 1
C. it outputs 2
D. it outputs 3

Correct Answer: A
Explanation

Explanation/Reference:
Similar to Q119. However the correct way to access managled name "`o._Class__foo()`" are replaced with "`o.__Class_foo()`". Note the numbers of "_" before "Class" and "foo()" are differents.

Since the name managling attribute are not specified correct, error will occur e.g.:

```
Traceback (most recent call last):
  File "C:/Temp/test.py", line 8, in <module>
    o.__Class_foo()
AttributeError: 'Class' object has no attribute '__Class_foo'. Did you mean:
'_Class__foo'?
```

Code for showing the above error:

```
class Class:
~~~~Var = 0
~~~~def __foo(self):
~~~~~~~~Class.Var += 1
~~~~~~~~return Class.Var

o = Class()
o.__Class_foo()
print(o.__Class_foo())
```