* Python case sensitive when dealing with identifiers.
* Identifiers can be of any length.
* Variable names should not start with a number.
* As Python has no concept of private variables, leading underscores are used to indicate variables that must not be accessed from outside the class.
* Eval is not a keyword.
* Keywords in Python are True, False and None are capitalized while the others are in lower case.
* in is a keyword.
* When both of the operands are integer then python chops out the fraction part and gives you the round off value, to get the accurate answer use floor division. This is floor division. For ex, 5/2 = 2.5 but both of the operands are integer so answer of this expression in python is 2.To get the 2.5 answer, use floor division.
* For order of precedence, just remember this PEMDAS that is, Parenthesis, Exponentiation, Division, Multiplication, Addition, Subtraction. Note that the precedence order of Division and Multiplication is the same. Likewise, the order of Addition and Subtraction is also the same.
* Operators with the same precedence are evaluated in Left to Right manner.
* Python shell throws a NoneType object back.
* Return Type of function id is int.

What error occurs when you execute?

apple = mango

a) SyntaxError

b) NameError

c) ValueError

d) TypeError

B

* List data type can store any values within it.
* Dictionary stores values in terms of keys and values.
* l (or L) stands for long.
* Infinity is a special case of floating point numbers. It can be obtained by float(‘inf’).
* ~x is equivalent to -(x+1).
* Numbers starting with a 0 are octal numbers but 9 isn’t allowed in octal numbers.
* cmp(x, y) returns 1 if x > y, 0 if x == y and -1 if x < y.

Which of the following is incorrect?

a) float(‘inf’)

b) float(‘nan’)

c) float(’56’+’78’)

d) float(’12+34′)

D

What is the result of round(0.5) – round(-0.5)?

a) 1.0

b) 2.0

c) 0.0

d) None of the mentioned

B

* ^ is the Binary XOR operator.
* The order of precedence is: %, +.
* exponentiation operator (\*\*) has its associativity from right to left.

What is the value of the following expression?

2+4.00, 2\*\*4.0

a) (6.0, 16.0)

b) (6.00, 16.00)

c) (6, 16)

d) (6.00, 16.0)

A

This is because the result is automatically rounded off to one decimal place.

* // is the operator for truncation division. It it called so because it returns only the integer part of the quotient, truncating the decimal part.
* associativity of \*\* operator is from right to left.
* Coercion is the implicit (automatic) conversion of operands to a common type. Coercion is automatically performed on mixed-type expressions. The expression 1.7 % 2 is evaluated as 1.7 % 2.0 (that is, automatic conversion of int to float).
* Type conversion is nothing but explicit conversion of operands to a specific type.
* The ^ operator represent bitwise XOR operation. &: bitwise AND, | : bitwise OR and ! represents bitwise NOT.

To find the decimal value of 1111, that is 15, we can use the function:

a) int(1111,10)

b) int(‘1111’,10)

c) int(1111,2)

d) int(‘1111’,2)

D

* The expression int(‘1111’,2) gives the result 15. The expression int(‘1111’, 10) will give the result 1111.

Which of the following expressions results in an error?

a) int(1011)

b) int(‘1011’,23)

c) int(1011,2)

d) int(‘1011’)

C

The expression int(1011,2) results in an error. Had we written this expression as int(‘1011’,2), then there would not be an error.

Which of the following expressions results in an error?

a) float(‘10’)

b) int(‘10’)

c) float(’10.8’)

d) int(’10.8’)

D

All of the above examples show explicit conversion. However the expression int(’10.8’) results in an error.

It is not possible for the two’s complement value to be equal to the original value in any case. State whether this statement is true or false.

a) True

b) False

B

In most cases the value of two’s complement is different from the original value. However, there are cases in which the two’s complement value may be equal to the original value. For example, the two’s complement of 10000000 is also equal to 10000000. Hence the statement is false.

* Bitwise XOR gives 1 if either of the bits is 1 and 0 when both of the bits are 1.
* Any odd number on being AND-ed with 1 always gives 1. Any even number on being AND-ed with this value always gives 0.

What is the value of this expression:

bin(10-2)+bin(12^4)

a) 0b10000

b) 0b10001000

c) 0b1000b1000

d) 0b10000b1000

D

The output of bin(10-2)= 0b1000 and that of bin(12^4) is ob1000. Hence the output of the above expression is: 0b10000b1000.

* The Boolean function returns true if the argument passed to the bool function does not amount to zero.

What is the output of the snippet of code shown below?

['hello', 'morning'][bool('')]

a) error

b) no output

c) hello

d) morning

C

* The function not returns true if the argument amounts to false, and false if the argument amounts to true.

What is the output of the code shown below?

l=[1, 0, 2, 0, 'hello', '', []]  
list(filter(bool, l))

a) Error

b) [1, 0, 2, 0, ‘hello’, ”, []]

c) [1, 0, 2, ‘hello’, ”, []]

d) [1, 2, ‘hello’]

D

The code shown above returns a new list containing only those elements of the list l which do not amount to zero. Hence the output is: [1, 2, ‘hello’]

The output of the snippet of code shown below is:

'%d %s %g you' %(1, 'hello', 4.0)

a) Error

b) 1 hello you 4.0

c) 1 hello 4 you

d) 1 4 hello you

C

In the snippet of code shown above, three values are inserted into the target string. When we insert more than one value, we should group the values on the right in a tuple. The % formatting expression operator expects either a single item or a tuple of one or more items on its right side.

What is the output of the code shown?

x=3.3456789  
'%f | %e | %g' %(x, x, x)

a) Error

b) ‘3.3456789 | 3.3456789+00 | 3.345678’

c) ‘3.345678 | 3.345678e+0 | 3.345678’

d) ‘3.345679 | 3.345679e+00 | 3.34568’

D

The %f %e and %g format specifiers represent floating point numbers in different ways. %e and %E are the same, except that the exponent is in lowercase. %g chooses the format by number content. Hence the output of this code is: ‘3.345679 | 3.345679e+00 | 3.34568’.

* The built in function vars() returns a dictionary containing all the variables that exist in the place.

The output of the code shown below is:

s='{0}, {1}, and {2}'  
s.format('hello', 'good', 'morning')

a) ‘hello good and morning’

b) ‘hello, good, morning’

c) ‘hello, good, and morning’

d) Error

C

Within the subject string, curly braces designate substitution targets and arguments to be inserted either by position or keyword. Hence the output of the code shown above:’hello, good,and morning’.

The output of the two codes shown below is the same. State whether true or false.

'{0:.2f}'.format(1/3.0)  
'%.2f'%(1/3.0)

a) True

b) False

A

The two codes shown above represent the same operation, but in different formats. The output of both of these functions is: ‘0.33’. Hence the statement is true.

l=list('HELLO')  
'first={0[0]}, third={0[2]}'.format(l)

a) ‘first=H, third=L’

b) ‘first=0, third=2’

c) Error

d) ‘first=0, third=L’

A

In the code shown above, the value for first is substituted by l[0], that is H and the value for third is substituted by l[2], that is L. Hence the output of the code shown above is: ‘first=H, third=L’. The list l= [‘H’, ‘E’, ‘L’, ‘L’, ‘O’].

l=list('HELLO')  
p=l[0], l[-1], l[1:3]  
'a={0}, b={1}, c={2}'.format(\*p)

a) Error

b) “a=’H’, b=’O’, c=(E, L)”

c) “a=H, b=O, c=[‘E’, ‘L’]”

d) Junk value

C

In the code shown above, the value for a is substituted by l[0], that is ‘H’, the value of b is substituted by l[-1], that is ‘O’ and the value for c is substituted by l[1:3]. Here the use of \*p is to unpack a tuple’s items into individual function arguments.

The formatting method {1:<10} represents the \_\_\_\_\_\_\_\_\_\_\_ positional argument, \_\_\_\_\_\_\_\_\_ justified in a 10 character wide field.

a) first, right

b) second, left

c) first, left

d) second, right

B

The formatting method {1:<10} represents the second positional argument, left justified in a 10 character wide field.

hex(255), int('FF', 16), 0xFF

a) [0xFF, 255, 16, 255]

b) (‘0xff’, 155, 16, 255)

c) Error

d) (‘0xff’, 255, 255)

D

The code shown above converts the value 255 into hexadecimal, that is, 0xff. The value ‘FF’ into integer. Hence the output of the code shown is: (‘0xff’, 255, 255).

The output of the two codes shown below is the same. State whether this statement is true or false.

bin((2\*\*16)-1)  
 '{}'.format(bin((2\*\*16)-1))

a) True

b) False

A

The output of both of the codes shown above is ‘0b1111111111111111’.

'{a}{b}{a}'.format(a='hello', b='world')

a) ‘hello world’

b) ‘hello’ ‘world’ ‘hello’

c) ‘helloworldhello’

d) ‘hello’ ‘hello’ ‘world’

C

What is the output of the following?

x = ['ab', 'cd']  
**for** i **in** x:  
 i.upper()  
**print**(x)

a) [‘ab’, ‘cd’].

b) [‘AB’, ‘CD’].

c) [None, None].

d) none of the mentioned

View Answer

Answer: a

Explanation: The function upper() does not modify a string in place, it returns a new string which isn’t being stored anywhere.

* range(str) is not allowed.

What is the output of the following?

**for** i **in** range(0):  
 **print**(i)

a) 0

b) no output

c) error

d) none of the mentioned

View Answer

Answer: b

Explanation: range(0) is empty.

**for** i **in** [1, 2, 3, 4][::-1]:  
 **print** (i)

a) 1 2 3 4

b) 4 3 2 1

c) error

d) none of the mentioned

View Answer

Answer: b

Explanation: [::-1] reverses the list.

What is the output of the following?

x = (i **for** i **in** range(3))  
**for** i **in** x:  
 **print**(i)  
**for** i **in** x:  
 **print**(i)

a) 0 1 2

b) error

c) 0 1 2 0 1 2

d) none of the mentioned

View Answer

Answer: a

Explanation: We can loop over a generator object only once.

* not in isn’t allowed in for loops.
* Values cannot be modified in the case of tuple, that is, tuple is immutable
* Tuples are immutable and don’t have an append method. An exception is thrown in this case.

What is the data type of (1)?

a) Tuple

b) Integer

c) List

d) Both tuple and integer

View Answer

Answer: b

Explanation: A tuple of one element must be created as (1,).

* Tuple slicing exists
* ‘tuple’ object doesn’t support item deletion.

Is the following line of code valid?

>>> a,b=1,2,3

a) Yes, this is an example of tuple unpacking. a=1 and b=2

b) Yes, this is an example of tuple unpacking. a=(1,2) and b=3

c) No, too many values to unpack

d) Yes, this is an example of tuple unpacking. a=1 and b=(2,3)

View Answer

Answer: c

Explanation: For unpacking to happen, the number of values of the right hand side must be equal to the number of variables on the left hand side.

Tuples can’t be made keys of a dictionary. True or False?

a) True

b) False

View Answer

Answer: b

Explanation: Tuples can be made keys of a dictionary because they are hashable.