What does the following code do? def a(b, c, d): pass	Answer: defines a function, which does nothing
What gets printed? Assuming python version 3.x print(type(1/2))	Answer: class 'float' / type 'int' in 2.7
What is the output of the following code? print(type([1,2]))	Answer: class 'list' / type 'list' in 2.7
What gets printed? def f(): pass print(type(f()))	Answer: class 'NoneType' / type 'NoneType' in 2.7
What should the below code print? print(type(1J))	Answer: class 'complex' / type 'complex' in 2.7
What is the output of the following code? print(type(lambda:None))	Answer: class 'function' / type 'function' in 2.7
What is the output of the below program? a = [1,2,3,None,(),[],] print(len(a))	Answer: 6
What gets printed? d = lambda p: p * 2 t = lambda p: p * 3 x = 2 x = d(x) x = t(x) x = d(x) print(x)	Answer: 24
What gets printed? x = 4.5 y = 2 print(x//y) print(x/y)	Answer: 2.0 2.25
What gets printed? nums = set([1,1,2,3,3,3,4]) print(len(nums))	Answer: 4

What gets printed?
 	
x = True
y = False
z = False
if x or y and z:
  print("yes")
else:
  print("no")
	Answer: "yes"

What gets printed?
 	
x = True
y = False
z = False
if not x or y:
  print 1
elif not x or not y and z:
  print 2
elif not x or y or not y and x:
  print 3
else:
  print 4
	Answer: 3

If PYTHONPATH is set in the environment, which directories are searched for modules? A) PYTHONPATH directory B) current directory C) home directory D) installation dependent default path	Answer: A, B, D Explanation: First is the current directory, then is the PYTHONPATH directory if set, then is the installation dependent default path
What gets printed? print(r"\nwoow")	Answer: \nwoow
What gets printed? print('elo' 'elo')	Answer: eloelo
What gets printed? print("\x48\x49!")	Answer: HI!
What gets printed? print(0xA + 0xa)	Answer: 20

What gets printed?
 	
class parent:
  def __init__(self, param):
  self.v1 = param
 	
class child(parent):
  def __init__(self, param):
  self.v2 = param
 	
obj = child(11)
print("{} {}".format(obj.v1, obj.v2))
	Answer: AttributeError: child instance has no attribute 'v1'. self.v1 was never created as a variable since the parent __init__ was not explicitly called

What gets printed? k1 = {'e', 'd'} k2 = {'e':1} print(type(k1), type(k2))	Answer: class 'set', class 'dict'

What gets printed?
 	
class Account:
  def __init__(self, id):
  self.id = id
  id = 666
 	
acc = Account(123)
print(acc.id)
	Answer: 123

What gets printed? name = 'snow storm' print(name[6:8])	Answer: 'to'
Which is the correct operator for power(x^y)? a) X^y b) X**y c) X^^y d) None of the mentioned	Answer: b Explanation: In python, power operator is x**y i.e. 2**3=8.
Which one of these is floor division? a) / b) // c) % d) None of the mentioned	Answer: b Explanation: 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.
What is the order of precedence in python? i) Parentheses ii) Exponential iii) Division iv) Multiplication v) Addition vi) Subtraction a) i,ii,iii,iv,v,vi b) ii,i,iii,iv,v,vi c) ii,i,iv,iii,v,vi d) i,ii,iii,iv,vi,v	Answer: a Explanation: For order of precedence, just remember this PEDMAS.
What is answer of this expression, 22 % 3 is? a) 7 b) 1 c) 0 d) 5	Answer: b Explanation: Modulus operator gives remainder. So, 22%3 gives 1 remainder.
You can perform mathematical operation on String? a) True b) False	Answer: b Explanation: You can’t perform mathematical operation on string even if string looks like integers.
Operators with the same precedence are evaluated in which manner? a) Left to Right b) Right to Left	Answer: a Explanation: None.
What is the output of this expression, 3*1**3? a) 27 b) 9 c) 3 d) 1	Answer: c Explanation: First this expression will solve 1**3 because exponential have higher precedence than multiplication, so 1**3 = 1 and 3*1 = 3. Final answer is 3.
Which one of the following have the same precedence? a) Addition and Subtraction b) Multiplication and Division c) a and b d) None of the mentioned	Answer: c Explanation: None.
Int(x) means variable x is converted to integer. a) True b) False	Answer: a Explanation: None.
Which one of the following have the highest precedence in the expression? a) Exponential b) Addition c) Multiplication d) Parentheses	Answer: d Explanation: None.
Is Python case sensitive when dealing with identifiers? a) yes b) no c) sometimes only d) none of the mentioned	Answer: a Explanation: Case is always significant.
What is the maximum possible length of an identifier? a) 31 characters b) 63 characters c) 79 characters d) none of the mentioned	Answer: d Explanation: Identifiers can be of any length.
Which of the following is invalid? a) _a = 1 b) __a = 1 c) __str__ = 1 d) none of the mentioned	Answer: d Explanation: All the statements will execute successfully but at the cost of reduced readability.
Which of the following is an invalid variable? a) my_string_1 b) 1st_string c) foo d) _	Answer: b Explanation: Variable names should not start with a number.
Why are local variable names beginning with an underscore discouraged? a) they are used to indicate a private variables of a class b) they confuse the interpreter c) they are used to indicate global variables d) they slow down execution	Answer: a Explanation: As Python has no concept of private variables, leading underscores are used to indicate variables that must not be accessed from outside the class.
Which of the following is not a keyword? a) eval b) assert c) nonlocal d) pass	Answer: a Explanation: eval can be used as a variable.
All keywords in Python are in a) lower case b) UPPER CASE c) Capitalized d) none	Answer: d Explanation: True, False and None are capitalized while the others are in lower case.
Which of the following is true for variable names in Python? a) unlimited length b) all private members must have leading and trailing underscores c) underscore and ampersand are the only two special charaters allowed d) none	Answer: a Explanation: Variable names can be of any length.
Which of the following is an invalid statement? a) abc = 1,000,000 b) a b c = 1000 2000 3000 c) a,b,c = 1000, 2000, 3000 d) a_b_c = 1,000,000	Answer: b Explanation: Spaces are not allowed in variable names.
Which of the following cannot be a variable? a) __init__ b) in c) it d) on	Answer: b Explanation: in is a keyword.
Which module in Python supports regular expressions? a) re b) regex c) pyregex d) none of the mentioned	Answer: a Explanation: re is a part of the standard library and can be imported using: import re.
Which of the following creates a pattern object? a) re.create(str) b) re.regex(str) c) re.compile(str) d) re.assemble(str)	Answer: c Explanation: It converts a given string into a pattern object.
What does the function re.match do? a) matches a pattern at the start of the string b) matches a pattern at any position in the string c) such a function does not exist d) none of the mentioned	Answer: a Explanation: It will look for the pattern at the beginning and return None if it isn’t found.
What does the function re.search do? a) matches a pattern at the start of the string b) matches a pattern at any position in the string c) such a function does not exist d) none of the mentioned	Answer: b Explanation: It will look for the pattern at any position in the string.

What is the output of the following? sentence = 'we are humans' matched = re.match(r'(.*) (.*?) (.*)', sentence) print(matched.groups()) a) (‘we’, ‘are’, ‘humans’) b) (we, are, humans) c) (‘we’, ‘humans’) d) ‘we are humans’	Answer: a Explanation: This function returns all the subgroups that have been matched.
What is the output of the following? sentence = 'we are humans' matched = re.match(r'(.*) (.*?) (.*)', sentence) print(matched.group()) a) (‘we’, ‘are’, ‘humans’) b) (we, are, humans) c) (‘we’, ‘humans’) d) ‘we are humans’	Answer: d Explanation: This function returns the entire match.
What is the output of the following? sentence = 'we are humans' matched = re.match(r'(.*) (.*?) (.*)', sentence) print(matched.group(2)) a) ‘are’ b) ‘we’ c) ‘humans’ d) ‘we are humans’	Answer: c Explanation: This function returns the particular subgroup.
What is the output of the following? sentence = 'horses are fast' regex = re.compile('(?P<animal>\w+) (?P<verb>\w+) (?P<adjective>\w+)') matched = re.search(regex, sentence) print(matched.groupdict()) a) {‘animal’: ‘horses’, ‘verb’: ‘are’, ‘adjective’: ‘fast’} b) (‘horses’, ‘are’, ‘fast’) c) ‘horses are fast’ d) ‘are’	Answer: a Explanation: This function returns a dictionary that contains all the mathches.
What is the output of the following? sentence = 'horses are fast' regex = re.compile('(?P<animal>\w+) (?P<verb>\w+) (?P<adjective>\w+)') matched = re.search(regex, sentence) print(matched.groups()) a) {‘animal’: ‘horses’, ‘verb’: ‘are’, ‘adjective’: ‘fast’} b) (‘horses’, ‘are’, ‘fast’) c) ‘horses are fast’ d) ‘are’	Answer: b Explanation: This function returns all the subgroups that have been matched.
What is the output of the following? sentence = 'horses are fast' regex = re.compile('(?P<animal>\w+) (?P<verb>\w+) (?P<adjective>\w+)') matched = re.search(regex, sentence) print(matched.group(2)) a) {‘animal’: ‘horses’, ‘verb’: ‘are’, ‘adjective’: ‘fast’} b) (‘horses’, ‘are’, ‘fast’) c) ‘horses are fast’ d) ‘are’	Answer: d Explanation: This function returns the particular subgroup.
What is the output of print 0.1 + 0.2 == 0.3? a) True b) False c) Machine dependent d) Error	Answer: b Explanation: Neither of 0.1, 0.2 and 0.3 can be represented accurately in binary. The round off errors from 0.1 and 0.2 accumulate and hence there is a difference of 5.5511e-17 between (0.1 + 0.2) and 0.3.
Which of the following is not a complex number? a) k = 2 + 3j b) k = complex(2, 3) c) k = 2 + 3l d) k = 2 + 3J	Answer: c Explanation: l (or L) stands for long.
What is the type of inf? a) Boolean b) Integer c) Float d) Complex	Answer: c Explanation: Infinity is a special case of floating point numbers. It can be obtained by float(‘inf’).
What does ~4 evaluate to? a) -5 b) -4 c) -3 d) +3	Answer: a Explanation: ~x is equivalent to -(x+1).
What does ~~~~~~5 evaluate to? a) +5 b) -11 c) +11 d) -5	Answer: a Explanation: ~x is equivalent to -(x+1).
Which of the following is incorrect? a) x = 0b101 b) x = 0x4f5 c) x = 19023 d) x = 03964	Answer: d Explanation: Numbers starting with a 0 are octal numbers but 9 isn’t allowed in octal numbers.
What is the result of cmp(3, 1)? a) 1 b) 0 c) True d) False	Answer: a Explanation: cmp(x, y) returns 1 if x > y, 0 if x == y and -1 if x < y.
What is the output of the following? x = ['ab', 'cd'] for i in x: x.append(i.upper()) print(x) a) [‘AB’, ‘CD’] b) [‘ab’, ‘cd’, ‘AB’, ‘CD’] c) [‘ab’, ‘cd’] d) none of the mentioned	Answer: d Explanation: The loop does not terminate as new elements are being added to the list in each iteration.

What is the output of the following?
i = 1
while True:
  if i%3 == 0:
  break
  print(i)
  i + = 1
a) 1 2
b) 1 2 3
c) error
d) none of the mentioned
	Answer: c
	Explanation: SyntaxError, there shouldn’t be a space between + and = in +=.

What is the output of the following?
i = 1
while True:
  if i%0O7 == 0:
  break
  print(i)
  i += 1
 	
a) 1 2 3 4 5 6
b) 1 2 3 4 5 6 7
c) error
d) none of the mentioned
	Answer: a
	Explanation: Control exits the loop when i becomes 7.

What is the output of the following?
i = 5
while True:
  if i%0O11 == 0:
  break
  print(i)
  i += 1
a) 5 6 7 8 9 10
b) 5 6 7 8
c) 5 6
d) error
	Answer: b
	Explanation: 0O11 is an octal number.

What is the output of the following?
i = 5
while True:
  if i%0O9 == 0:
  break
  print(i)
  i += 1
a) 5 6 7 8
b) 5 6 7 8 9
c) 5 6 7 8 9 10 11 12 13 14 15 ….
d) error
	Answer: d
	Explanation: 9 isn't allowed in an octal number.

What is the output of the following?
i = 1
while True:
  if i%2 == 0:
  break
  print(i)
  i += 2
a) 1
b) 1 2
c) 1 2 3 4 5 6 …
d) 1 3 5 7 9 11 …
	Answer: d
	Explanation: The loop does not terminate since i is never an even number.

What is the output of the following?
i = 2
while True:
  if i%3 == 0:
  break
  print(i)
  i += 2
a) 2 4 6 8 10 …
b) 2 4
c) 2 3
d) error
	Answer: b
	Explanation: The numbers 2 and 4 are printed. The next value of i is 6
	which is divisible by 3 and hence control exits the loop.

What is the output of the following?
i = 1
while False:
  if i%2 == 0:
  break
  print(i)
  i += 2
a) 1
b) 1 3 5 7 …
c) 1 2 3 4 …
d) none of the mentioned
	Answer: d
	Explanation: Control does not enter the loop because of False.

What is the output of the following?
True = False
while True:
  print(True)
  break
a) True
b) False
c) None
d) none of the mentioned
	Answer: d
	Explanation: SyntaxError, True is a keyword and it's value cannot be changed.

What is the output of the following?
i = 0
while i < 3:
  print(i)
  i += 1
else:
  print(0)
a) 0 1 2 3 0
b) 0 1 2 0
c) 0 1 2
c) error
	Answer: b
	Explanation: The else part is executed when the condition in the while statement is false.

What is the output of the following? x = 'abcdef' while i in x: print(i, end=' ') a) a b c d e f b) abcdef c) i i i i i i … d) error	Answer: d Explanation: NameError, i is not defined.
What is the output of the following? x = 'abcdef' i = 'i' while i in x: print(i, end=' ') a) no output b) i i i i i i … c) a b c d e f d) abcdef	Answer: a Explanation: "i" is not in "abcdef".
What is the output of the following? x = 'abcdef' i = 'a' while i in x: print(i, end = ' ') a) no output b) i i i i i i … c) a a a a a a … d) a b c d e f	Answer: c Explanation: As the value of i or x isn't changing, the condition will always evaluate to True.
What is the output of the following? x = 'abcdef' i = 'a' while i in x: print('i', end = ' ') a) no output b) i i i i i i … c) a a a a a a … d) a b c d e f	Answer: b Explanation: As the value of i or x isn't changing, the condition will always evaluate to True.

What is the output of the following?
x = 'abcdef'
i = 'a'
while i in x:
  x = x[:-1]
  print(i, end = ' ')
a) i i i i i i
b) a a a a a a
c) a a a a a
d) none of the mentioned
	Answer: b
	Explanation: The string x is being shortened by one charater in each iteration.

What is the output of the following? x = 'abcdef' i = 'a' while i in x[:-1]: print(i, end = ' ') a) a a a a a b) a a a a a a c) a a a a a a … d) a	Answer: c Explanation: String x is not being altered and i is in x[:-1].

What is the output of the following?
x = 'abcdef'
i = 'a'
while i in x:
  x = x[1:]
  print(i, end = ' ')
a) a a a a a a
b) a
c) no output
d) error
	Answer: b
	Explanation: The string x is being shortened by one charater in each iteration.

What is the output of the following? x = 'abcdef' i = 'a' while i in x[1:]: print(i, end = ' ') a) a a a a a a b) a c) no output d) error	Answer: c Explanation: i is not in x[1:].
What is the output of the following? x = 'abcd' for i in x: print(i.upper()) a) a b c d b) A B C D c) a B C D d) error	Answer: b Explanation: The instance of the string returned by upper() is being printed.
What is the output of the following? x = 'abcd' for i in range(x): print(i) a) a b c d b) 0 1 2 3 c) error d) none of the mentioned	Answer: c Explanation: range(str) is not allowed.
What is the output of the following? x = 'abcd' for i in range(len(x)): print(i) a) a b c d b) 0 1 2 3 c) error d) none of the mentioned	Answer: b Explanation: i takes values 0, 1, 2 and 3.
What is the output of the following? x = 'abcd' for i in range(len(x)): print(i.upper()) a) a b c d b) 0 1 2 3 c) error d) none of the mentioned	Answer: c Explanation: Objects of type int have no attribute upper().
What is the output of the following? x = 'abcd' for i in range(len(x)): i.upper() print (x) a) a b c d b) 0 1 2 3 c) error d) none of the mentioned	Answer: c Explanation: Objects of type int have no attribute upper().
What is the output of the following? x = 'abcd' for i in range(len(x)): x[i].upper() print (x) a) abcd b) ABCD c) error d) none of the mentioned	Answer: a Explanation: Changes do not happen in-place, rather a new instance of the string is returned.
What is the output of the following? x = 'abcd' for i in range(len(x)): i[x].upper() print (x) a) abcd b) ABCD c) error d) none of the mentioned	Answer: c Explanation: Objects of type int aren't subscriptable.

What is the output of the following?
x = 'abcd'
for i in range(len(x)):
  x = 'a'
  print(x)
a) a
b) abcd abcd abcd
c) a a a a
d) none of the mentioned
	Answer: c
	Explanation: range() is computed only at the time of entering the loop.

What is the output of the following?
x = 'abcd'
for i in range(len(x)):
  print(x)
  x = 'a'
a) a
b) abcd abcd abcd abcd
c) a a a a
d) none of the mentioned
	Answer: d
	Explanation: abcd a a a is the output as x is modified only after 'abcd' has been printed once.

What is the output of the following? d = {0: 'a', 1: 'b', 2: 'c'} for i in d: print(i) a) 0 1 2 b) a b c c) 0 a 1 b 2 c d) none of the mentioned	Answer: a Explanation: Loops over the keys of the dictionary.
What is the output of the following? d = {0: 'a', 1: 'b', 2: 'c'} for x, y in d: print(x, y) a) 0 1 2 b) a b c c) 0 a 1 b 2 c d) none of the mentioned	Answer: d Explanation: Error, objects of type int aren't iterable.
What is the output of the following? d = {0: 'a', 1: 'b', 2: 'c'} for x, y in d.items(): print(x, y) a) 0 1 2 b) a b c c) 0 a 1 b 2 c d) none of the mentioned	Answer: c Explanation: Loops over key, value pairs.
What is the output of the following? d = {0: 'a', 1: 'b', 2: 'c'} for x in d.keys(): print(d[x]) a) 0 1 2 b) a b c c) 0 a 1 b 2 c d) none of the mentioned	Answer: b Explanation: Loops over the keys and prints the values.
What is the output of the following? d = {0: 'a', 1: 'b', 2: 'c'} for x in d.values(): print(x) a) 0 1 2 b) a b c c) 0 a 1 b 2 c d) none of the mentioned	Answer: b Explanation: Loops over the values.

What is the output of the following? d = {0: 'a', 1: 'b', 2: 'c'} for x in d.values(): print(d[x]) a) 0 1 2 b) a b c c) 0 a 1 b 2 c d) none of the mentioned	Answer: d Explanation: Causes a KeyError.
What is the output of the following? d = {0, 1, 2} for x in d.values(): print(x) a) 0 1 2 b) None None None c) error d) none of the mentioned	Answer: c Explanation: Objects of type set have no attribute values.
What is the output of the following? d = {0, 1, 2} for x in d: print(x) a) 0 1 2 b) {0, 1, 2} {0, 1, 2} {0, 1, 2} c) error d) none of the mentioned	Answer: a Explanation: Loops over the elements of the set and prints them.
What is the output of the following? d = {0, 1, 2} for x in d: print(d.add(x)) a) 0 1 2 b) 0 1 2 0 1 2 0 1 2 … c) None None None d) none of the mentioned	Answer: c Explanation: Variable x takes the values 0, 1 and 2. set.add() returns None which is printed.
What is the output of the following? for i in range(0): print(i) a) 0 b) (nothing is printed) c) error d) none of the mentioned	Answer: b Explanation: range(0) is empty.
What is the output of the following? for i in range(int(2.0)): print(i) a) 0.0 1.0 b) 0 1 c) error d) none of the mentioned	Answer: b Explanation: range(int(2.0)) is the same as range(2).
What is the output of the following? for i in range(float('inf')): print (i) a) 0.0 0.1 0.2 0.3 … b) 0 1 2 3 … c) 0.0 1.0 2.0 3.0 … d) none of the mentioned	Answer: d Explanation: Error, objects of type float cannot be interpreted as an integer.
What is the output of the following? for i in range(int(float('inf'))): print (i) a) 0.0 0.1 0.2 0.3 … b) 0 1 2 3 … c) 0.0 1.0 2.0 3.0 … d) none of the mentioned	Answer: d Explanation: OverflowError, cannot convert float infinity to integer.
What is the output of the following? 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	Answer: b Explanation: [::-1] reverses the list.
What is the output of the following? for i in ''.join(reversed(list('abcd'))): print (i) a) a b c d b) d c b a c) error d) none of the mentioned	Answer: b Explanation: ”.join(reversed(list(‘abcd’))) reverses a string.
What is the output of the following? for i in 'abcd'[::-1]: print (i) a) a b c d b) d c b a c) error d) none of the mentioned	Answer: b Explanation: [::-1] reverses the string.
What is the output of the following? for i in '': print (i) a) None b) (nothing is printed) c) error d) none of the mentioned	Answer: b Explanation: The string does not have any character to loop over.

What is the output of the following?
x = 2
for i in range(x):
  x += 1
  print (x)
a) 0 1 2 3 4 …
b) 0 1
c) 3 4
d) 0 1 2 3
	Answer: c
	Explanation: Variable x is incremented and printed twice.

What is the output of the following?
x = 2
for i in range(x):
  x -= 2
  print (x)
a) 0 1 2 3 4 …
b) 0 -2
c) 0
d) error
	Answer: b
	Explanation: The loop is entered twice.

What is the output of the following?
for i in range(5):
  if i == 5:
  break
  else:
  print(i)
else:
  print('Here')
a) 0 1 2 3 4 Here
b) 0 1 2 3 4 5 Here
c) 0 1 2 3 4
d) 1 2 3 4 5
	Answer: a
	Explanation: The else part is executed if control doesn’t break out of the loop.

What is the output of the following? x = (i for i in range(3)) for i in x: print(i) a) 0 1 2 b) error c) 0 1 2 0 1 2 d) none of the mentioned	Answer: a Explanation: The first statement creates a generator object.

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

What is the output of the following? string = 'my name is x' for i in string: print (i, end=', ') a) m, y, , n, a, m, e, , i, s, , x, b) m, y, , n, a, m, e, , i, s, , x c) my, name, is, x, d) error	Answer: a Explanation: Variable i takes the value of one character at a time.
What is the output of the following? string = 'my name is x' for i in string.split(): print (i, end=', ') a) m, y, , n, a, m, e, , i, s, , x, b) m, y, , n, a, m, e, , i, s, , x c) my, name, is, x, d) error	Answer: c Explanation: Variable i takes the value of one word at a time.
What is the output of the following? a = [0, 1, 2, 3] for a[-1] in a: print(a[-1]) a) 0 1 2 3 b) 0 1 2 2 c) 3 3 3 3 d) error	Answer: b Explanation: The value of a[-1] changes in each iteration.
What is the output of the following? a = [0, 1, 2, 3] for a[0] in a: print(a[0]) a) 0 1 2 3 b) 0 1 2 2 c) 3 3 3 3 d) error	Answer: a Explanation: The value of a[0] changes in each iteration. Since the first value that it takes is itself, there is no visible error in the current example.

What is the output of the following?
a = [0, 1, 2, 3]
i = -2
for i not in a:
  print(i)
  i += 1
a) -2 -1
b) 0
c) error
d) none of the mentioned
	Answer: c
	Explanation: SyntaxError, not in isn't allowed in for loops.

What is the output of the following? string = 'my name is x' for i in ' '.join(string.split()): print (i, end=', ') a) m, y, , n, a, m, e, , i, s, , x, b) m, y, , n, a, m, e, , i, s, , x c) my, name, is, x, d) error	Answer: a Explanation: Variable i takes the value of one character at a time.
What is the output of the following? print('abc. DEF'.capitalize()) a) abc def b) ABC DEF c) Abc def d) Abc Def	Answer: c Explanation: The first letter of the string is converted to uppercase and the others are converted to lowercase.
What is the output of the following? print('abcdef'.center()) a) cd b) abcdef c) error d) none of the mentioned	Answer: c Explanation: The function center() takes atleast one parameter.
What is the output of the following? print('abcdef'.center(0)) a) cd b) abcdef c) error d) none of the mentioned	Answer: c Explanation: The entire string is printed when the argument passed to center() is less than the length of the string.
What is the output of the following? print('*', 'abcdef'.center(7), '*') a) * abcdef * b) * abcdef * c) *abcdef * d) * abcdef*	Answer: b Explanation: Padding is done towards the left-hand-side first when the final string is of odd length. Extra spaces are present since we haven't overridden the value of sep.
What is the output of the following? print('*', 'abcdef'.center(7), '*', sep='') a) * abcdef * b) * abcdef * c) *abcdef * d) * abcdef*	Answer: d Explanation: Padding is done towards the left-hand-side first when the final string is of odd length.
What is the output of the following? print('*', 'abcde'.center(6), '*', sep='') a) * abcde * b) * abcde * c) *abcde * d) * abcde*	Answer: c Explanation: Padding is done towards the right-hand-side first when the final string is of even length.
What is the output of the following? print('abcdef'.center(7, 1)) a) 1abcdef b) abcdef1 c) abcdef d) error	Answer: d Explanation: TypeError, the fill character must be a character, not an int.
What is the output of the following? print('abcdef'.center(7, '1')) a) 1abcdef b) abcdef1 c) abcdef d) error	Answer: a Explanation: The character '1' is used for padding instead of a space.
What is the output of the following? print('abcdef'.center(10, '12')) a) 12abcdef12 b) abcdef1212 c) 1212abcdef d) error	Answer: d Explanation: The fill character must be exactly one character long.
What is the output of the following? print('xyyzxyzxzxyy'.count('yy', 1)) a) 2 b) 0 c) 1 d) none of the mentioned	Answer: a Explanation: Counts the number of times the substring 'yy' is present in the given string, starting from position 1.
What is the output of the following? print('xyyzxyzxzxyy'.count('yy', 2)) a) 2 b) 0 c) 1 d) none of the mentioned	Answer: c Explanation: Counts the number of times the substring 'yy' is present in the given string, starting from position 2.

What is the output of the following? print('xyyzxyzxzxyy'.count('xyy', 0, 100)) a) 2 b) 0 c) 1 d) error	Answer: a Explanation: An error will not occur if the end value is greater than the length of the string itself.
What is the output of the following? print('xyyzxyzxzxyy'.count('xyy', 2, 11)) a) 2 b) 0 c) 1 d) error	Answer: b Explanation: Counts the number of times the substring ‘xyy’ is present in the given string, starting from position 2 and ending at position 11.
What is the output of the following? print('xyyzxyzxzxyy'.count('xyy', -10, -1)) a) 2 b) 0 c) 1 d) error	Answer: b Explanation: Counts the number of times the substring ‘xyy’ is present in the given string, starting from position 2 and ending at position 11.
What is the output of the following? print('abc'.encode()) a) abc b) ‘abc’ c) b’abc’ d) h’abc’	Answer: c Explanation: A bytes object is returned by encode.
What is the default value of encoding in encode()? a) ascii b) qwerty c) utf-8 d) utf-16	Answer: c Explanation: The default value of encoding is utf-8.
What is the output of the following? print('xyyzxyzxzxyy'.endswith('xyy')) a) 1 b) True c) 3 d) 2	Answer: b Explanation: The function returns True if the given string ends with the specified substring.
What is the output of the following? print('xyyzxyzxzxyy'.endswith('xyy', 0, 2)) a) 0 b) 1 c) True d) False	Answer: d Explanation: The function returns False if the given string does not end with the specified substring.
What is the output of the following? print('abcdef'.find('cd')) a) True b) 2 c) 3 d) none of the mentioned	Answer: b Explanation: The first position in the given string at which the substring can be found is returned.
What is the output of the following? print('ccdcddcd'.find('c')) a) 4 b) 0 c) error d) True	Answer: b Explanation: The first position in the given string at which the substring can be found is returned.
What is the output of the following? print('Hello {0} and {1}'.format('foo', 'bin')) a) Hello foo and bin b) Hello {0} and {1} foo bin c) error d) Hello 0 and 1	Answer: a Explanation: The numbers 0 and 1 represent the position at which the strings are present.
What is the output of the following? print('Hello {1} and {0}'.format('bin', 'foo')) a) Hello foo and bin b) Hello bin and foo c) error d) none of the mentioned	Answer: a Explanation: The numbers 0 and 1 represent the position at which the strings are present.
What is the output of the following? print('Hello {} and {}'.format('foo', 'bin')) a) Hello foo and bin b) Hello {} and {} c) error d) Hello and	Answer: a Explanation: It is the same as Hello {0} and {1}.
What is the output of the following? print('Hello {name1} and {name2}'.format('foo', 'bin')) a) Hello foo and bin b) Hello {name1} and {name2} c) error d) Hello and	Answer: c Explanation: The arguments passed to the function format aren’t keyword arguments.
What is the output of the following? print('Hello {0!r} and {0!s}'.format('foo', 'bin')) a) Hello foo and bin b) Hello ‘foo’ and bin c) Hello foo and ‘bin’ d) error	Answer: b Explanation: !r causes the charactes ‘ or ” to be printed as well.
What is the output of the following? print('Hello {0} and {1}'.format(('foo', 'bin'))) a) Hello foo and bin b) Hello (‘foo’, ‘bin’) and (‘foo’, ‘bin’) c) error d) none of the mentioned	Answer: c Explanation: IndexError, the tuple index is out of range.
What is the output of the following? print('Hello {0[0]} and {0[1]}'.format(('foo', 'bin'))) a) Hello foo and bin b) Hello (‘foo’, ‘bin’) and (‘foo’, ‘bin’) c) error d) none of the mentioned	Answer: a Explanation: The elements of the tuple are accessed by their indices.
What is the output of the following? print('The sum of {0} and {1} is {2}'.format(2, 10, 12)) a) The sum of 2 and 10 is 12 b) error c) The sum of 0 and 1 is 2 d) none of the mentioned	Answer: a Explanation: The arguments passed to the function format can be integers also.
What is the output of the following? print('The sum of {0:b} and {1:x} is {2:o}'.format(2, 10, 12)) a) The sum of 2 and 10 is 12 b) The sum of 10 and a is 14 c) The sum of 10 and a is c d) error	Answer: b Explanation: 2 is converted to binary, 10 to hexadecimal and 12 to octal.
What is the output of the following? print('{:,}'.format(1112223334)) a) 1,112,223,334 b) 111,222,333,4 c) 1112223334 d) error	Answer: a Explanation: A comma is added after every third digit from the right.
What is the output of the following? print('{:,}'.format('1112223334')) a) 1,112,223,334 b) 111,222,333,4 c) 1112223334 d) error	Answer: d Explanation: An integer is expected.

What is the output of the following? print('{:$}'.format(1112223334)) a) 1,112,223,334 b) 111,222,333,4 c) 1112223334 d) error	Answer: d Explanation: $ is an invalid format code.
What is the output of the following? print('{:#}'.format(1112223334)) a) 1,112,223,334 b) 111,222,333,4 c) 1112223334 d) error	Answer: c Explanation: The number is printed as it is.
What is the output of the following? print('{0:.2%}'.format(1/3)) a) 0.33 b) 0.33% c) 33.33% d) 33%	Answer: c Explanation: The symbol % is used to represent the result of an expression as a percentage.
What is the output of the following? print('ab12'.isalnum()) a) True b) False c) None d) error	Answer: a Explanation: The string has only letters and digits.
What is the output of the following? print('ab,12'.isalnum()) a) True b) False c) None d) error	Answer: b Explanation: The character , is not a letter or a digit.
What is the output of the following? print('ab'.isalpha()) a) True b) False c) None d) error	Answer: a Explanation: The string has only letters.
What is the output of the following? print('a B'.isalpha()) a) True b) False c) None d) error	Answer: b Explanation: Space is not a letter.
What is the output of the following? print('0xa'.isdigit()) a) True b) False c) None d) error	Answer: b Explanation: Hexadecimal digits aren't considered as digits (a-f).
What is the output of the following? print(''.isdigit()) a) True b) False c) None d) error	Answer: b Explanation: If there are no characters then False is returned.
What is the output of the following? print('my_string'.isidentifier()) a) True b) False c) None d) error	Answer: a Explanation: It is a valid identifier.
What is the output of the following? print('__foo__'.isidentifier()) a) True b) False c) None d) error	Answer: a Explanation: It is a valid identifier.
What is the output of the following? print('abc'.islower()) a) True b) False c) None d) error	Answer: a Explanation: There are no uppercase letters.
What is the output of the following? print('a@ 1,'.islower()) a) True b) False c) None d) error	Answer: a Explanation: There are no uppercase letters.
What is the output of the following? print('11'.isnumeric()) a) True b) False c) None d) error	Answer: a Explanation: All the character are numeric.
What is the output of the following? print('1.1'.isnumeric()) a) True b) False c) None d) error	Answer: b Explanation: The character . is not a numeric character.
What is the output of the following? print('1@ a'.isprintable()) a) True b) False c) None d) error	Answer: a Explanation: All those characters are printable.
What is the output of the following? print(''' '''.isspace()) a) True b) False c) None d) error	Answer: a Explanation: A newline character is considered as space.
What is the output of the following? print('\t'.isspace()) a) True b) False c) None d) error	Answer: a Explanation: Tabspaces are considered as spaces.
What is the output of the following? print('HelloWorld'.istitle()) a) True b) False c) None d) error	Answer: b Explanation: The letter W is uppercased.
What is the output of the following? print('Hello World'.istitle()) a) True b) False c) None d) Error	Answer: a Explanation: It is in title form.

What is the output of the following? print('1Rn@'.lower()) a) n b) 1rn@ c) rn d) r	Answer: b Explanation: Uppercase letters are converted to lowercase. The other characters are left unchanged.
What is the output of the following? print(''' \tfoo'''.lstrip()) a) \tfoo b) foo c) foo d) none of the mentioned	Answer: b Explanation: All leading whitespace is removed.
What is the output of the following? print('xyyzxxyxyy'.lstrip('xyy')) a) error b) zxxyxyy c) z d) zxxy	Answer: b Explanation: The leading characters containing xyy are removed.
What is the output of the following? print('xyxxyyzxxy'.lstrip('xyy')) a) zxxy b) xyxxyyzxxy c) xyxzxxy d) none of the mentioned	Answer: a Explanation: All combinations of the characters passed as an argument are removed from the left hand side.
What is the output of the following? print('cba'.maketrans('abc', '123')) a) {97: 49, 98: 50, 99: 51} b) {65: 49, 66: 50, 67: 51} c) 321 d) 123	Answer: a Explanation: A translation table is returned by maketrans.
What is the output of the following? print('a'.maketrans('ABC', '123')) a) {97: 49, 98: 50, 99: 51} b) {65: 49, 66: 50, 67: 51} c) {97: 49} d) 1	Answer: a Explanation: maketrans() is a static method so it's behaviour does not depend on the object from which it is being called.
What is the output of the following? print('abcdef'.partition('cd')) a) ('ab', 'ef') b) ('abef') c) ('ab', 'cd', 'ef') d) 2	Answer: c Explanation: The string is split into three parts by partition.
What is the output of the following? print('abcdefcdgh'.partition('cd')) a) ('ab', 'cd', 'ef', 'cd', 'gh') b) ('ab', 'cd', 'efcdgh') c) ('abcdef', 'cd', 'gh') d) error	Answer: b Explanation: The string is partitioned at the point where the separator first appears.
What is the output of the following? print('abcd'.partition('cd')) a) ('ab', 'cd', '') b) ('ab', 'cd') c) error d) none of the mntioned	Answer: a Explanation: The last item is a null string.
What is the output of the following? print('abef'.partition('cd')) a) ('abef') b) ('abef', 'cd', '') c) ('abef', '', '') d) error	Answer: c Explanation: The separator is not present in the string hence the second and the third elements of the tuple are null strings.
What is the output of the following? print('abcdef12'.replace('cd', '12')) a) ab12ef12 b) abcdef12 c) ab12efcd d) none of the mentioned	Answer: a Explanation: All occurences of the first substring are replaced by the second substring.
What is the output of the following? print('abef'.replace('cd', '12')) a) abef b) 12 c) error d) none of the mentioned	Answer: a Explanation: The first substring is not present in the given string and hence nothing is replaced.
What is the output of the following? print('abcefd'.replace('cd', '12')) a) ab1ef2 b) abcefd c) ab1efd d) ab12ed2	Answer: b Explanation: The first substring is not present in the given string and hence nothing is replaced.
What is the output of the following? print('xyyxyyxyxyxxy'.replace('xy', '12', 0)) a) xyyxyyxyxyxxy b) 12y12y1212x12 c) 12yxyyxyxyxxy d) xyyxyyxyxyx12	Answer: a Explanation: The first 0 occurances of the given substring are replaced.
What is the output of the following? print('xyyxyyxyxyxxy'.replace('xy', '12', 100)) a) xyyxyyxyxyxxy b) 12y12y1212x12 c) none of the mentioned d) error	Answer: b Explanation: The first 100 occurances of the given substring are replaced.
What is the output of the following? print('abcdefcdghcd'.split('cd')) a) ['ab', 'ef', 'gh'] b) ['ab', 'ef', 'gh', ''] c) ('ab', 'ef', 'gh') d) ('ab', 'ef', 'gh', '')	Answer: b Explanation: The given string is split and a list of substrings is returned.
What is the output of the following? print('abcdefcdghcd'.split('cd', 0)) a) ['abcdefcdghcd'] b) 'abcdefcdghcd' c) error d) none of the mentioned	Answer: a Explanation: The given string is split at 0 occurances of the specified substring.
What is the output of the following? print('abcdefcdghcd'.split('cd', -1)) a) ['ab', 'ef', 'gh'] b) ['ab', 'ef', 'gh', ''] c) ('ab', 'ef', 'gh') d) ('ab', 'ef', 'gh', '')	Answer: b Explanation: Calling the function with a negative value for maxsplit is the same as calling it without any maxsplit specified. The string will be split into as many substring s as possible.
What is the output of the following? print('ab\ncd\nef'.splitlines()) a) ['ab', 'cd', 'ef'] b) ['ab\n', 'cd\n', 'ef\n'] c) ['ab\n', 'cd\n', 'ef'] d) ['ab', 'cd', 'ef\n']	Answer: a Explanation: It is similar to calling split('\n').
What is the output of the following? print('Ab!2'.swapcase()) a) AB!@ b) ab12 c) aB!2 d) aB1@	Answer: c Explanation: Lowercase letters are converted to uppercase and vice-versa.

What is the output of the following? print('ab cd ef'.title()) a) Ab cd ef b) Ab cd eF c) Ab Cd Ef d) none of the mentioned	Answer: c Explanation: The first letter of every word is capitalized.
What is the output of the following? print('ab cd-ef'.title()) a) Ab cd-ef b) Ab Cd-ef c) Ab Cd-Ef d) none of the mentioned	Answer: c Explanation: The first letter of every word is capitalized. Special symbols terminate a word.
What is the output of the following? print('abcd'.translate('a'.maketrans('abc', 'bcd'))) a) bcde b) abcd c) error d) none of the mentioned	Answer: d Explanation: The output is bcdd since no translation is provided for d.
What is the output of the following? print('abcd'.translate({97: 98, 98: 99, 99: 100})) a) bcde b) abcd c) error d) none of the mentioned	Answer: d Explanation: The output is bcdd since no translation is provided for d.
What is the output of the following? print('abcd'.translate({'a': '1', 'b': '2', 'c': '3', 'd': '4'})) a) abcd b) 1234 c) error d) none of the mentioned	Answer: a Explanation: The function translate expects a dictionary of integers. Use maketrans() instead of doing the above.
What is the output of the following? print('ab'.zfill(5)) a) 000ab b) 00ab0 c) 0ab00 d) ab000	Answer: a Explanation: The string is padded with zeroes on the left hand side. It is useful for formatting numbers.
What is the output of the following? print('+99'.zfill(5)) a) 00+99 b) 00099 c) +0099 d) +++99	Answer: c Explanation: Zeroes are filled in between the first sign and the rest of the string.
Which is/are the basic I/O connections in file? a) Standard Input b) Standard Output c) Standard Errors d) All of the above e) None of the above	Answer: d Explanation: Standard input, standard output and standard error. Standard input is the data that goes to the program. The standard input comes from a keyboard. Standard output is where we print our data with the print keyword. Unless redirected, it is the terminal console. The standard error is a stream where programs write their error messages. It is usually the text terminal.

What is the output of this program?
import sys
print 'Enter your name: '
name = ''
while True:
c = sys.stdin.read(1)
if c == '\n':
break
	Answer: a
	Explanation: In order to work with standard I/O streams, we must import the sys module.
	The read() method reads one character from the standard input.
	In our example we get a promt saying “Enter your name”. We enter our name and press enter.
	The enter key generates the new line character: \n.

What is the output of this program? import sys sys.stdout.write(' Hello\n')sys.stdout.write('Python\n') a) Compilation Error b) Runtime Error c) Hello Python d) Hello Python	Answer: d Explanation: None
Which of the following mode will refer to binary data? a) r b) w c) + d) b	Answer:d Explanation: Mode Meaning is as explained below:
What is the pickling? a) It is used for object serialization b) It is used for object deserialization c) None of the mentioned	Answer: a Explanation: Pickle is the standard mechanism for object serialization. Pickle uses a simple stack-based virtual machine that records the instructions used to reconstruct the object. This makes pickle vulnerable to security risks by malformed or maliciously constructed data,
What is unpickling? a) It is used for object serialization b) It is used for object deserialization c) None of the mentioned	Answer: b Explanation: We have been working with simple textual data. What if we are working with objects rather than simple text? For such situations, we can use the pickle module. This module serializes Python objects. The Python objects are converted into byte streams and written to
What is the correct syntax of open() function? a) file = open(file_name [, access_mode][, buffering]) b) file object = open(file_name [, access_mode][, buffering]) c) file object = open(file_name) d) None of the mentioned	Answer: b Explanation: Open() function correct syntax with the parameter details as shown below:
Correct syntax of file.writelines() is? a) file.writelines(sequence) b) fileObject.writelines() c) fileObject.writelines(sequence) d) None of the mentioned	Answer: c Explanation: The method writelines() writes a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings. There is no return value.

Following is the syntax for writelines() method:
fileObject.writelines(sequence).
10. Correct syntax of file.readlines() is?
a) fileObject.readlines(sizehint);
b) fileObject.readlines();
c) fileObject.readlines(sequence)
d) None of the mentioned
	Answer: a
	Explanation: The method readlines() reads until EOF using readline() and returns a list containing the lines.
	If the optional sizehint argument is present, instead of reading up to EOF,
	whole lines totalling approximately sizehint bytes (possibly after rounding up to an internal buffer size) are read.

In file handling, what does this terms means “r, a”? a) read, append b) append, read c) All of the mentioned d) None of the the mentioned	Answer: a Explanation: r- reading, a-appending.
What is the use of “w” in file handling? a) Read b) Write c) Append d) None of the the mentioned	Answer: b Explanation: This opens the file for writing. It will create the file if it doesn’t exist, and if it does, it will overwrite it.
What is the use of “a” in file handling? a) Read b) Write c) Append d) None of the the mentioned	Answer:c Explanation: This opens the fhe file in appending mode. That means, it will be open for writing and everything will be written to the end of the file.
fWhich function is used to read all the characters? a) Read() b) Readcharacters() c) Readall() d) Readchar()	Answer:a Explanation:The read function reads all characters fh = open(“filename”, “r”) content = fh.read(). 5. Which function is used to read single line from file? a) Readline() b) Readlines() c) Readstatement() d) Readfullline()

content = fh.readline(). 6. Which function is used to write all the characters? a) write() b) writecharacters() c) writeall() d) writechar()	Answer: a Explanation:To write a fixed sequence of characters to a file fh = open(“hello.txt”,”w”) write(“Hello World”). 7. Which function is used to write a list of string in a file a) writeline() b) writelines() c) writestatement() d) writefullline()
Whether we can create a text file in python? a) Yes b) No c) None of the mentioned	Answer: a Explanation: Yes we can create a file in python. Creation of file is as shown below.
file = open(“newfile.txt”, “w”) file.write(“hello world in the new file\n”) file.write(“and another line\n”) file.close(). 10. Which of the following is modes of both writing and reading in binary format in file.? a) wb+ b) w c) wb d) w+	Answer: a Explanation: Here is the description below
Which of the following is not a valid mode to open a file? a) ab b) rw c) r+ d) w+	Answer: b Explanation: Use r+, w+ or a+ to perform both read and write operations using a single file object.
What is the difference between r+ and w+ modes? a) no difference b) in r+ the pointer is initially placed at the beginning of the file and the pointer is at the end for w+ c) in w+ the pointer is initially placed at the beginning of the file and the pointer is at the end for r+	Answer: b Explanation: none.
How do you get the name of a file from a file object (fp)? a) fp.name b) fp.file(name) c) self.__name__(fp) d) fp.__name__()	Answer: a Explanation: name is an attribute of the file object.
Which of the following is not a valid attribute of a file object (fp)? a) fp.name b) fp.closed c) fp.mode d) fp.size	Answer: d Explanation: fp.size has not been implemented.
How do you close a file object (fp)? a) close(fp) b) fclose(fp) c) fp.close() d) fp.__close__()	Answer: c Explanation: close() is a method of the file object.
How do you get the current position within the file? a) fp.seek() b) fp.tell() c) fp.loc d) fp.pos	Answer: b Explanation: It gives the current position as an offset from the start of file.
How do you rename a file? a) fp.name = ‘new_name.txt’ b) os.rename(existing_name, new_name) c) os.rename(fp, new_name) d) os.set_name(existing_name, new_name)	Answer: b Explanation: os.rename() is used to rename files.
How do you delete a file? a) del(fp) b) fp.delete() c) os.remove(‘file’) d) os.delete(‘file’)	Answer: c Explanation: os.remove() is used to delete files.
How do you change the file position to an offset value from the start? a) fp.seek(offset, 0) b) fp.seek(offset, 1) c) fp.seek(offset, 2) d) none of the mentioned	Answer: a Explanation: 0 indicates that the offset is with respect to the start.
What happens if no arguments are passed to the seek function? a) file position is set to the start of file b) file position is set to the end of file c) file position remains unchanged d) error	Answer: d Explanation: seek() takes at least one argument.
a) 2 b) 3 c) The numbers are equal d) None of the mentioned	Answer: b Explanation: The maximum function returns the maximum of the parameters, in this case the numbers supplied to the function. It uses a simple if..else statement to find the greater value and then returns that value.
Which of the following is a features of DocString? a) Provide a convenient way of associating documentation with Python modules, functions, classes, and methods b) All functions should have a docstring c) Docstrings can be accessed by the __doc__ attribute on objects d) All of the mentioned	Answer: d Explanation: Python has a nifty feature called documentation strings, usually referred to by its shorter name docstrings. DocStrings are an important tool that you should make use of since it helps to document the program better and makes it easier to understand.
Which are the advantages of functions in python? a) Reducing duplication of code b) Decomposing complex problems into simpler pieces c) Improving clarity of the code d) Reuse of code e) Information hiding f) All of the mentioned	Answer: f Explanation: None.
What are the two types of functions? a) Custom function b) Built-in function c) User-Defined function d) System function	Answer: b and c Explanation: Built-in functions and user defined ones. The built-in functions are part of the Python language. Examples are: dir(), len() or abs(). The user defined functions are functions created with the def keyword.
Where is function defined? a) Module b) Class c) Another function d) None of the mentioned	Answer: a, b and c Explanation: Functions can be defined inside a module, a class or another function.
What is called when a function is defined inside a class? a) Module b) Class c) Another function d) Method	Answer: d Explanation: None.
Which of the following is the use of id() function in python? a) Id returns the identity of the object b) Every object doesn’t have a unique id c) All of the mentioned d) None of the mentioned	Answer: a Explanation: Each object in Python has a unique id. The id() function returns the object’s id.
Which of the following refers to mathematical function? a) sqrt b) rhombus c) add d) rhombus	Answer: a Explanation: Functions that are always available for usage, functions that are contained within external modules, which must be imported and functions defined by a programmer with the def keyword.
Eg: math import sqrt The sqrt() function is imported from the math module. 7. What is the output of below program? def cube(x): return x * x * x x = cube(3) print x a) 9 b) 3 c) 27 d) 30	Answer: c Explanation: A function is created to do a specific task. Often there is a result from such a task. The return keyword is used to return values from a function. A function may or may not return a value. If a function does not have a return keyword, it will send a none value.

What is the output of the below program? def C2F(c): return c * 9/5 + 32print C2F(100)print C2F(0) a) 212 32 b) 314 24 c) 567 98 d) None of the mentioned	Answer: a Explanation: None.
What is the output of the below program? def power(x, y=2): r = 1 for i in range(y): r = r * x return rprint power(3)print power(3, 3) a) 212 32 b) 9 27 c) 567 98 d) None of the mentioned	Answer: b Explanation: The arguments in Python functions may have implicit values. An implicit value is used, if no value is provided. Here we created a power function. The function has one argument with an implicit value. We can call the function with one or two arguments.

What is the output of the below program?
def sum(*args):
  "Function returns the sum
  of all values"'
  r = 0
  for i in args:
  r += i
  return r
 	
print sum.__doc__
	Answer: a
	Explanation: We use the * operator to indicate, that the function will accept arbitrary number of arguments.
	The sum() function will return the sum of all arguments.
	The first string in the function body is called the function documentation string. It is used to document the function.
	The string must be in triple quotes.

Python supports the creation of anonymous functions at runtime, using a construct called _____? a) Lambda b) pi c) anonymous d) None of the mentioned	Answer: a Python supports the creation of anonymous functions (i.e. functions that are not bound to a name) at runtime, using a construct called lambda. Lambda functions are restricted to a single expression. They can be used wherever normal functions can be used.
The lambda function is executed. The number 8 is passed to the anonymous function and it returns 48 as the result. Note that z is not a name for this function. It is only a variable to which the anonymous function was assigned. 3. What is the output of below program? lamb = lambda x: x ** 3print(lamb(5)) a) 15 b) 555	Answer: c Explanation: None.
Is Lambda contains return statements a) True b) False	Answer: b Explanation: lambda definition does not include a return statement — it always contains an expression which is returned. Also note that we can put a lambda definition anywhere a function is expected, and we don't have to assign
Lambda is a statement. a) True b) False	Answer: b Explanation: None.
Lambda contains block of statements a) True b) False	Answer: b Explanation: None.
What is the output of below program? def f(x, y, z): return x + y + z f(2, 30, 400) a) 432 b) 24000 c) 430 d) None of the mentioned	Answer: a Explanation: None.
What is the output of below program? def writer(): title = 'Sir' name = (lambda x:title + ' ' + x) return name who = writer()who('Arthur') a) Arthur Sir b) Sir Arthur c) Arthur d) None of the mentioned	Answer: b Explanation: None.
What is the output of this program? L = [lambda x: x ** 2, lambda x: x ** 3, lambda x: x ** 4] for f in L: print(f(3)) a) 27 81 343 b) 6 9 12 c) 9 27 81 d) None of the mentioned	Answer: c Explanation: None.
What is the output of this program? min = (lambda x, y: x if x < y else y) min(101*99, 102*98) a) 9997 b) 9999 c) 9996 d) None of the mentioned	Answer: c Explanation: None.
How many except statements can a try-except block have? a) zero b) one c) more than one d) more than zero	Answer: d Explanation: There has to be at least one except statement.
When will the else part of try-except-else be executed? a) always b) when an exception occurs c) when no exception occurs d) when an exception occurs in to except block	Answer: c Explanation: The else part is executed when no exception occurs.

Is the following code valid?
try:
  # Do something
except:
  # Do something
finally:
  # Do something
a) no, there is no such thing as finally
b) no, finally cannot be used with except
c) no, finally must come before except
d) yes
	Answer: b
	Explanation: Refer documentation.

Is the following code valid?
try:
  # Do something
except:
  # Do something
else:
  # Do something
 	
a) no, there is no such thing as else
b) no, else cannot be used with except
c) no, else must come before except
d) yes
	Answer: d
	Explanation: Refer documentation.

Can one block of except statements handle multiple exception? a) yes, like except TypeError, SyntaxError [,…] b) yes, like except [TypeError, SyntaxError] c) no d) none of the mentioned	Answer: a Explanation: Each type of exception can be specified directly. There is no need to put it in a list.
When is the finally block executed? a) when there is no exception b) when there is an exception c) only if some condition that has been specified is satisfied d) always	Answer: d Explanation: The finally block is always executed.

What is the output of the following code?
def foo():
  try:
  return 1
  finally:
  return 2
k = foo()
print(k)
a) 1
b) 2
c) 3
d) error, there is more than one return statement in a single try-finally block
	Answer: b
	Explanation: The finally block is executed even there is a return statement in the try block.

What is the output of the following code?
def foo():
  try:
  print(1)
  finally:
  print(2)
foo()
 	
a) 1 2
b) 1
c) 2
d) none of the mentioned
	Answer: a
	Explanation: No error occurs in the try block so 1 is printed. Then the finally block is executed and 2 is printed.

What is the output of the following?
try:
  if '1' != 1:
  raise 'someError'
  else:
  print('someError has not occured')
except 'someError':
  print ('someError has occured')
a) someError has occured
b) someError has not occured
c) invalid code
d) none of the mentioned
	Answer: c
	Explanation: A new exception class must inherit from a BaseException. There is no such inheritance here.

What happens when '1' == 1 is executed? a) we get a True b) we get a False c) an TypeError occurs d) a ValueError occurs	Answer: b Explanation: It simply evaluates to False and does not raise any exception.
What is the type of each element in sys.argv? a) set b) list c) tuple d) string	Answer: d Explanation: It is a list of strings.
What is the length of sys.argv? a) number of arguments b) number of arguments + 1 c) number of arguments – 1 d) none of the mentioned	Answer: b Explanation: The first argument is the name of the program itself. Therefore the length of sys.argv is one more than the number arguments.
What is the output of the following code? def foo(k): k[0] = 1 q = [0] foo(q) print(q) a) [0] b) [1] c) [1, 0] d) [0, 1]	Answer: b Explanation: Lists are passed by reference.
How are keyword arguments specified in the function heading? a) one star followed by a valid identifier b) one underscore followed by a valid identifier c) two stars followed by a valid identifier d) two underscores followed by a valid identifier	Answer: c Explanation: Refer documentation.
How many keyword arguments can be passed to a function in a single function call? a) zero b) one c) zero or more d) one or more	Answer: c Explanation: zero keyword arguments may be passed if all the arguments have default values.
What is the output of the following code? def foo(fname, val): print(fname(val)) foo(max, [1, 2, 3]) foo(min, [1, 2, 3]) a) 3 1 b) 1 3 c) error d) none of the mentioned	Answer: a Explanation: It is possible to pass function names as arguments to other functions.
What is the output of the following code? def foo(): return total + 1 total = 0 print(foo()) a) 0 b) 1 c) error d) none of the mentioned	Answer: b Explanation: It is possible to read the value of a global variable directly.

What is the output of the following code?
def foo():
  total += 1
  return total
total = 0
print(foo())
a) 0
b) 1
c) error
d) none of the mentioned
	Answer: c
	Explanation: It is not possible to change the value of a global variable without explicitly specifying it.

What is the output of the following code?
def foo(x):
  x = ['def', 'abc']
  return id(x)
q = ['abc', 'def']
print(id(q) == foo(q))
a) True
b) False
c) None
d) error
	Answer: b
	Explanation: A new object is created in the function.

What is the output of the following code?
def foo(i, x=[]):
  x.append(i)
  return x
for i in range(3):
  print(foo(i))
a) [0] [1] [2]
b) [0] [0, 1] [0, 1, 2]
c) [1] [2] [3]
d) [1] [1, 2] [1, 2, 3]
	Answer: b
	Explanation: When a list is a default value, the same list will be reused.

How are variable length arguments specified in the function heading? a) one star followed by a valid identifier b) one underscore followed by a valid identifier c) two stars followed by a valid identifier d) two underscores followed by a valid identifier	Answer: a Explanation: Refer documentation.
Which module in the python standard library parses options received from the command line? a) getopt b) os c) getarg d) main	Answer: a Explanation: getopt parses options received from the command line.

What is the type of sys.argv? a) set b) list c) tuple d) string	Answer: b Explanation: It is a list of elements.
What is the value stored in sys.argv[0]? a) null b) you cannot access it c) the program's name d) the first argument	Answer: c Explanation: Refer documentation.
How are default arguments specified in the function heading? a) identifier followed by an equal to sign and the default value b) identifier followed by the default value within backticks (“) c) identifier followed by the default value within square brackets ([]) d) identifier	Answer: a Explanation: Refer documentation.
How are required arguments specified in the function heading? a) identifier followed by an equal to sign and the default value b) identifier followed by the default value within backticks (“) c) identifier followed by the default value within square brackets ([]) d) identifier	Answer: d Explanation: Refer documentation.
What is the output of the following code? def foo(x): x[0] = ['def'] x[1] = ['abc'] return id(x) q = ['abc', 'def'] print(id(q) == foo(q)) a) True b) False c) None d) error	Answer: a Explanation: The same object is modified in the function.
Where are the arguments recived from the command line stored? a) sys.argv b) os.argv c) argv d) none of the mentioned	Answer: a Explanation: Refer documentation.

What is the output of the following?
def foo(i, x=[]):
x.append(x.append(i))
return x
for i in range(3):
y = foo(i)
print(y)
a) [[[0]], [[[0]], [1]], [[[0]], [[[0]], [1]], [2]]]
b) [[0], [[0], 1], [[0], [[0], 1], 2]]
c) [[0], None, [1], None, [2], None]
d) [[[0]], [[[0]], [1]], [[[0]], [[[0]], [1]], [2]]]
	Answer: c
	Explanation: append() returns None.

What is the output of print(k) in the following? k = [print(i) for i in my_string if i not in 'aeiou'] print(k) a) all characters of my_string that aren't vowels b) a list of Nones c) list of Trues d) list of Falses	Answer: b Explanation: print() returns None.
What is the output of the following? my_string = 'hello world' k = [(i.upper(), len(i)) for i in my_string] print(k) a) [(‘HELLO’, 5), (‘WORLD’, 5)] b) [(‘H’, 1), (‘E’, 1), (‘L’, 1), (‘L’, 1), (‘O’, 1), (‘ ‘, 1), (‘W’, 1), (‘O’, 1), (‘R’, 1), (‘L’, 1), (‘D’, 1)] c) [(‘HELLO WORLD’, 11)] d) none of the mentioned	Answer: b Explanation: We are iterating over each letter in the string.

Which of the following is the correct expansion of list_1 = [expr(i) for i in list_0 if func(i)] ?
a)
list_1 = []
for i in list_0:
if func(i):
list_1.append(i)
 	
b)
for i in list_0:
if func(i):
list_1.append(expr(i))
c)
list_1 = []
for i in list_0:
if func(i):
list_1.append(expr(i))
d) none of the mentioned
	Answer: c
	Explanation: We have to create an empty list,
	loop over the contents of the existing list and check if a condtion is satisfied
	before performing some operation and adding it to the new list.

What is the output of the following? x = [i**+1 for i in range(3)]; print(x); a) [0, 1, 2] b) [1, 2, 5] c) error, **+ is not a valid operator d) error, ‘;’ is not allowed	Answer: a Explanation: i**+1 is evaluated as (i)**(+1).
What is the output of the following? print([i.lower() for i in 'HELLO']) a) [‘h’, ‘e’, ‘l’, ‘l’, ‘o’] b) ‘hello’ c) [‘hello’] d) hello	Answer: a Explanation: We are iterating over each letter in the string.
What is the output of the following? print([i+j for i in 'abc' for j in 'def']) a) [‘da’, ‘ea’, ‘fa’, ‘db’, ‘eb’, ‘fb’, ‘dc’, ‘ec’, ‘fc’] b) [[‘ad’, ‘bd’, ‘cd’], [‘ae’, ‘be’, ‘ce’], [‘af’, ‘bf’, ‘cf’]] c) [[‘da’, ‘db’, ‘dc’], [‘ea’, ‘eb’, ‘ec’], [‘fa’, ‘fb’, ‘fc’]] d) [‘ad’, ‘ae’, ‘af’, ‘bd’, ‘be’, ‘bf’, ‘cd’, ‘ce’, ‘cf’]	Answer: d Explanation: If it were to be executed as a nested for loop, i would be the outer loop and j the inner loop.
What is the output of the following? print([[i+j for i in 'abc'] for j in 'def']) a) [‘da’, ‘ea’, ‘fa’, ‘db’, ‘eb’, ‘fb’, ‘dc’, ‘ec’, ‘fc’] b) [[‘ad’, ‘bd’, ‘cd’], [‘ae’, ‘be’, ‘ce’], [‘af’, ‘bf’, ‘cf’]] c) [[‘da’, ‘db’, ‘dc’], [‘ea’, ‘eb’, ‘ec’], [‘fa’, ‘fb’, ‘fc’]] d) [‘ad’, ‘ae’, ‘af’, ‘bd’, ‘be’, ‘bf’, ‘cd’, ‘ce’, ‘cf’]	Answer: b Explanation: The inner list is generated once for each value of j.
What is the output of the following? print([if i%2==0: i; else: i+1; for i in range(4)]) a) [0, 2, 2, 4] b) [1, 1, 3, 3] c) error d) none of the mentioned	Answer: c Explanation: Syntax error.
Which of the following is the same as list(map(lambda x: x**-1, [1, 2, 3]))? a) [x**-1 for x in [(1, 2, 3)]] b) [1/x for x in [(1, 2, 3)]] c) [1/x for x in (1, 2, 3)] d) error	Answer: c Explanation: x**-1 is evaluated as (x)**(-1).

Which of the following will print True?
a = foo(2)
b = foo(3)
print(a < b)
a)
class foo:
def __init__(self, x):
self.x = x
def __lt__(self, other):
if self.x < other.x:
return False
else:
return True
 	
b)
class foo:
def __init__(self, x):
self.x = x
def __less__(self, other):
if self.x > other.x:
return False
else:
return True
c)
class foo:
def __init__(self, x):
self.x = x
def __lt__(self, other):
if self.x < other.x:
return True
else:
return False
d)
class foo:
def __init__(self, x):
self.x = x
def __less__(self, other):
if self.x < other.x:
return False
else:
return True
	Answer: c
	Explanation: __lt__ overloads the < operator.

Which operator is overloaded by __lg__()? a) < b) > c) != d) none of the mentioned	Answer: d Explanation: __lg__() is invalid.
Which function overloads the >> operator? a) __more__() b) __gt__() c) __ge__() d) none of the mentioned	Answer: d Explanation: __rshift__() overloads the >> operator.
Let A and B be objects of class Foo. Which functions are called when print(A + B) is executed? a) __add__(), __str__() b) __str__(), __add__() c) __sum__(), __str__() d) __str__(), __sum__()	Answer: a Explanation: The function __add__() is called first since it is within the bracket. The function __str__() is then called on the object that we recieved after adding A and B.
Which operator is overloaded by the __or__() function? a) || b) | c) // d) /	Answer: b Explanation: The function __or__() overloads the bitwise OR operator |.
Which function overloads the // operator? a) __div__() b) __ceildiv__() c) __floordiv__() d) __truediv__()	Answer: c Explanation: __floordiv__() is for //.
What the does random.seed(3) return? a) True b) None c) 3 d) 1	Answer: b Explanation: The function random.seed() always returns a None.
Which of the following cannot be returned by random.randrange(4)? a) 0 b) 3 c) 2.3 d) none of the mentioned	Answer: c Explanation: Only integers can be returned.
Which of the following is equivalent to random.randrange(3)? a) range(3) b) random.choice(range(0, 3)) c) random.shuffle(range(3)) d) random.select(range(3))	Answer: b Explanation: It returns one number from the given range.
The function random.randint(4) can return only one of the following values. Which? a) 4 b) 3.4 c) error d) none of the mentioned	Answer: c Explanation: Error, the function takes two arguments.
Which of the following is equivalent to random.randint(3, 6)? a) random.choice([3, 6]) b) random.randrange(3, 6) c) 3 + random.randrange(3) d) 3 + random.randrange(4)	Answer: d Explanation: random.randint(3, 6) can return any one of 3, 4, 5 and 6.
Which of the following will not be returned by random.choice(“1 ,”)? a) 1 b) (space) c) , d) none of the mentioned	Answer: d Explanation: Any of the characters present in the string may be returned.
Which of the following will never be displayed on executing print(random.choice({0: 1, 2: 3}))? a) 0 b) 1 c) KeyError: 1 d) none of the mentioned	Answer: a Explanation: It will not print 0 but dict[0] i.e. 1 may be printed.
What does random.shuffle(x) do when x = [1, 2, 3]? a) return a list in which the elements 1, 2 and 3 are in random positions b) do nothing, it is a placeholder for a function that is yet to be implemented c) shuffle the elements of the list in-place d) none of the mentioned	Answer: c Explanation: The elements of the list passed to it are shuffled in-place.
Which type of elements are accepted by random.shuffle()? a) strings b) lists c) tuples d) integers	Answer: b Explanation: Strings and tuples are immutable and an integer has no len().
What is the range of values that random.random() can return? a) [0.0, 1.0] b) (0.0, 1.0] c) (0.0, 1.0) d) [0.0, 1.0)	Answer: d Explanation: Any number that is greater than or equal to 0.0 and lesser than 1.0 can be returned.
What is returned by math.ceil(3.4)? a) 3 b) 4 c) 4.0 d) 3.0	Answer: b Explanation: The ceil function returns the smallest integer that is bigger than or equal to the number itself.
What is the value returned by math.floor(3.4)? a) 3 b) 4 c) 4.0 d) 3.0	Answer: a Explanation: The floor function returns the biggest number that is smaller than or equal to the number itself.

What is the output of print(math.copysign(3, -1))? a) 1 b) 1.0 c) -3 d) -3.0	Answer: d Explanation: The copysign function returns a float whose absolute value is that of the first argument and the sign is that of the second argument.
What is displayed on executing print(math.fabs(-3.4))? a) -3.4 b) 3.4 c) 3 d) -3	Answer: b Explanation: A negative floating point number is returned as a positive floating point number.
Is the function abs() same as math.fabs()? a) sometimes b) always c) never d) none of the mentioned	Answer: a Explanation: math.fabs() always returns a float and does not work with complex numbers whereas the return type of abs() is determined by the type of value that is passed to it.
What is the value returned by math.fact(6)? a) 720 b) 6 c) [1, 2, 3, 6] d) error	Answer: d Explanation: NameError, fact() is not defined.
What is the value of x if x = math.factorial(0)? a) 0 b) 1 c) error d) none of the mentioned	Answer: b Explanation: Factorial of 0 is 1.
What is math.factorial(4.0)? a) 24 b) 1 c) error d) none of the mentioned	Answer: a Explanation: The factorial of 4 is returned.
What is the output of print(math.factorial(4.5))? a) 24 b) 120 c) error d) none of the mentioned	Answer: c Explanation: Factorial is only defined for non-negative integers.
What is math.floor(0o10)? a) 8 b) 10 c) 0 d) 9	Answer: a Explanation: 0o10 is 8 and floor(8) is 8.
What does the function math.frexp(x) return? a) a tuple containing of the mantissa and the exponent of x b) a list containing of the mantissa and the exponent of x c) a tuple containing of the mantissa of x d) a list containing of the exponent of x	Answer: a Explanation: It returns a tuple with two elements. The first element is the mantissa and the second element is the exponent.
What is the result of math.fsum([.1 for i in range(20)])? a) 2.0 b) 20 c) 2 d) 2.0000000000000004	Answer: a Explanation: The function fsum returns an accurate floating point sum of the elements of its argument.
What is the result of sum([.1 for i in range(20)])? a) 2.0 b) 20 c) 2 d) 2.0000000000000004	Answer: d Explanation: There is some loss of accuracy when we use sum with floating point numbers. Hence the function fsum is preferable.
What is returned by math.isfinite(float('inf'))? a) True b) False c) None d) error	Answer: b Explanation: float('inf') is not a finite number.
What is returned by math.isfinite(float('nan'))? a) True b) False c) None d) error	Answer: b Explanation: float('nan') is not a finite number.
What is x if x = math.isfinite(float('0.0'))? a) True b) False c) None d) error	Answer: a Explanation: float('0.0') is a finite number.
What is the result of the following? >>> -float('inf') + float('inf') a) inf b) nan c) 0 d) 0.0	Answer: b Explanation: The result of float('inf')-float('inf') is undefined.
What is the output of the following? print(math.isinf(float('-inf'))) a) error, the minus sign shouldn't havve been inside the brackets b) error, there is no function called isinf c) True d) False	Answer: c Explanation: -float('inf') is the same as float('-inf').
What is the value of x if x = math.ldexp(0.5, 1)? a) 1 b) 2.0 c) 0.5 d) none of the mentioned	Answer: d Explanation: The value returned by ldexp(x, y) is x * (2 ** y). In the current case x is 1.0.
What is returned by math.modf(1.0)? a) (0.0, 1.0) b) (1.0, 0.0) c) (0.5, 1) d) (0.5, 1.0)	Answer: a Explanation: The first element is the fractional part and the second element is the integral part of the argument.
What is the result of math.trunc(3.1)? a) 3.0 b) 3 c) 0.1 d) 1	Answer: b Explanation: The integral part of the floating point number is returned.
What is the output of print(math.trunc('3.1'))? a) 3 b) 3.0 c) error d) none of the mentioned	Answer: c Explanation: TypeError, a string does not have __trunc__ method.
Which of the following is the same as math.exp(p)? a) e ** p b) math.e ** p c) p ** e d) p ** math.e	Answer: b Explanation: math.e is the constant defined in the math module.
What is returned by math.expm1(p)? a) (math.e ** p) – 1 b) math.e ** (p – 1) c) error d) none of the mentioned	Answer: a Explanation: One is subtracted from the result of math.exp(p) and returned.
What is the default base used when math.log(x) is found? a) e b) 10 c) 2 d) none of the mentioned	Answer: a Explanation: The natural log of x is returned by default.
Which of the following aren't defined in the math module? a) log2() b) log10() c) logx() d) none of the mentioned	Answer: c Explanation: log2() and log10() are defined in the math module.

What is returned by int(math.pow(3, 2))? a) 6 b) 9 c) error, third argument required d) error, too many arguments	Answer: b Explanation: math.pow(a, b) returns a ** b.
What is output of print(math.pow(3, 2))? a) 9 b) 9.0 c) None d) none of the mentioned	Answer: b Explanation: math.pow() returns a floating point number.
What is the value of x if x = math.sqrt(4)? a) 2 b) 2.0 c) (2, -2) d) (2.0, -2.0)	Answer: b Explanation: The function returns one floating point number.
What does math.sqrt(X, Y) do? a) calculate the Xth root of Y b) calculate the Yth root of X c) error d) return a tuple with the square root of X and Y	Answer: c Explanation: The function takes only one argument.
What does os.name contain? a) the name of the operating system dependent module imported b) the address of the module os c) error, it should've been os.name() d) none of the mentioned	Answer: a Explanation: It contains the name of the operating system dependent module imported such as 'posix', 'java' etc.
What does print(os.geteuid()) print? a) the group id of the current process b) the user id of the current process c) both the group id and the user of the current process d) none of the mentioned	Answer: b Explanation: os.geteuid() gives the user id while the os.getegid() gives the group id.
What does os.getlogin() return? a) name of the current user logged in b) name of the superuser c) gets a form to login as a different user d) all of the above	Answer: a Explanation: It returns the name of the user who is currently logged in and is running the script.
What does os.close(f) do? a) terminate the process f b) terminate the process f if f is not responding c) close the file descriptor f d) return an integer telling how close the file pointer is to the end of file	Answer: c Explanation: When a file descriptor is passed as an argument to os.close() it will be closed.
What does os.fchmod(fd, mode) do? a) change permission bits of the file b) change permission bits of the directory c) change permission bits of either the file or the directory d) none of the mentioned	Answer: a Explanation: The arguments to the function are a file descriptor and the new mode.
Which of the following functions can be used to read data from a file using a file descriptor? a) os.reader() b) os.read() c) os.quick_read() d) os.scan()	Answer: b Explanation: None of the other functions exist.
Which of the following returns a string that represents the present working directory? a) os.getcwd() b) os.cwd() c) os.getpwd() d) os.pwd()	Answer: a Explanation: The function getcwd() (get current working directory) returns a string that represents th present working directory.
What does os.link() do? a) create a symbolic link b) create a hard link c) create a soft link d) none of the mentioned	Answer: b Explanation: os.link(source, destination) will create a hard link from source to destination.
Which of the following can be used to create a directory? a) os.mkdir() b) os.creat_dir() c) os.create_dir() d) os.make_dir()	Answer: a Explanation: The function mkdir() creates a directory in the path specified.
Which of the following can be used to create a symbolic link? a) os.symlink() b) os.symb_link() c) os.symblin() d) os.ln()	Answer: a Explanation: It is the function that allows you to create a symbolic link.
What is the output of the following? print(999+1 is 1000) print(1+1 is 2) "	Answer: False, True Explanation: Python maintains a pool of objects representing the first few hundred integers and reuses them to save on memory and object creation. To make it even more confusing, the definition of what ""small integer"" is differs across Python versions. A mitigation here is to never use the 'is' operator for value comparison. The is operator is designed to deal exclusively with object identities.
What is the output of the following? print(2.2 * 3.0 == 3.3 * 2.0)	Answer: False Explanation: The cause of the above phenomena is indeed a rounding error: (2.2 * 3.0).hex() '0x1.a666666666667p+2' (3.3 * 2.0).hex() '0x1.a666666666666p+2
What is the output of the following? print(10**1000000 float('infinity'))	Answer: False
What is the output of the following? class X(object): ... def __init__(self): ... self.__private = 1 ... def get_private(self): ... return self.__private ... def has_private(self): ... return hasattr(self, '__private') x = X() print(x.has_private()) print(x.get_private()) "	Answer: False, 1 Explanation: Python does not support object attributes hiding. But there is a workaround based on the feature of double underscored attributes mangling. Although changes to attribute names occur only to code, attributes names hardcoded into string constants remain unmodified. This may lead to confusing behavior when a double underscored attribute visibly ""hides"" from getattr()/ hasattr() functions."
What is the output of the following? class X(object): ... def __init__(self): ... self.__private = 1 x = X() print(x.__private) x.__private = 2 print(x.__private) print(hasattr(x, '__private'))	Answer: AttributeError: 'X' object has no attribute '__private' 2 True
What is Python?	Answer: Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes.
What are the rules for local and global variables in Python?	Answer: In Python, variables that are only referenced inside a function are implicitly global. If a variable is assigned a new value anywhere within the function's body, it's assumed to be a local.
How do I copy an object in Python?	Answer: In general, try copy.copy() or copy.deepcopy() for the general case. Not all objects can be copied, but most can. Some objects can be copied more easily. Dictionaries have a copy() method: newdict = olddict.copy()
How do I convert a number to a string? "	Answer: To convert, e.g., the number 144 to the string '144', use the built-in function str(). If you want a hexadecimal or octal representation, use the built-in functions hex() or oct().
How do I convert between tuples and lists?	Answer: the function tuple(seq) converts any sequence (actually, any iterable) into a tuple with the same items in the same order.
What's a negative index?	Answer: Python sequences are indexed with positive numbers and negative numbers. For positive numbers 0 is the first index 1 is the second index and so forth.
What is a class?	Answer: A class is the particular object type created by executing a class statement. Class objects are used as templates to create instance objects,
How do I call a method defined in a base class from a derived class that overr	Answer: If you're using new-style classes, use the built-in super() function: class Derived(Base):
How can I organize my code to make it easier to change the base class?	Answer: You could define an alias for the base class, assign the real base class to it before your class definition,

Where is the math.py (socket.py, regex.py, etc.) source file?	Answer: There are (at least) three kinds of modules in Python: 1. modules written in Python (.py); 2. modules written in C and dynamically loaded (.dll, .pyd, .so, .sl, etc);
What is self?	Answer: Self is merely a conventional name for the first argument of a method. A method defined as meth(self, a, b, c) should be called as x.meth(a, b, c) for some instance x of the class
How do I apply a method to a sequence of objects? "	Answer: Use a list comprehension: result = [obj.method() for obj in List] More generically, you can try the following function: def method_map(objects, method, arguments): """method_map([a,b], "meth", (1,2)) gives [a.meth(1,2), b.meth(1,2)]""" nobjects = len(objects) methods = map(getattr, objects, [method]*nobjects) return map(apply, methods, [arguments]*nobjects)"
Why don't my signal handlers work?	Answer: The most common problem is that the signal handler is declared with the wrong argument list. It is called as handler(signum, frame) so it should be declared with two arguments: def handler(signum, frame): ...
How can I execute arbitrary Python statements from C?	Answer: the highest-level function to do this is PyRun_SimpleString() which takes a single string argument to be executed
What is Freeze for Windows?	Answer: Freeze is a program that allows you to ship a Python program as a single stand-alone executable file.
How do I interface to C++ objects from Python? "	Answer: Depending on your requirements, there are many approaches. To do this manually, begin by reading the ""Extending and Embedding"" document.
How do I generate random numbers in Python?	Answer: The standard module random implements a random number generator. Usage is simple: import random random.random() - This returns a random floating point number in the range [0, 1)

"What will be the output of the code below?
 	
def extendList(val, list=[]):
list.append(val)
return list
 	
list1 = extendList(10)
list2 = extendList(123,[])
list3 = extendList('a')
 	
print ""list1 = %s"" % list1
print ""list2 = %s"" % list2
print ""list3 = %s"" % list3
 	
How would you modify the definition of extendList to produce the presumably desired behavior?"
	Answer:
	
	list1 = [10, 'a']
	list2 = [123]
	list3 = [10, 'a']
	
	Explanation: New default list is created only once when the function is defined,
	and that same list is then used subsequently whenever extendList is invoked without a list argument being specified.
	This is because expressions in default arguments are calculated when the function is defined, not when it's called.
	list1 and list3 are therefore operating on the same default list, whereas list2 is operating on a separate list that it created (by passing its own empty list as the value for the list parameter).
	
	def extendList(val, list=None):
	if list is None:
	list = []
	list.append(val)
	return list

What will be the output of the code below? def multipliers(): return [lambda x : i * x for i in range(4)] print([m(2) for m in multipliers()]) How would you modify the definition of multipliers to produce the presumably desired behavior?	Answer: [6, 6, 6, 6] Explanation: The reason for this is that Python's closures are late binding. This means that the values of variables used in closures are looked up at the time the inner function is called. So as a result, when any of the functions returned by multipliers() are called, the value of i is looked up in the surrounding scope at that time. By then, regardless of which of the returned functions is called, the for loop has completed and i is left with its final value of 3. Therefore, every returned function multiplies the value it is passed by 3, so since a value of 2 is passed in the above code, they all return a value of 6 (i.e., 3 x 2). def multipliers(): for i in range(4): yield lambda x : i * x def multipliers(): return [lambda x, i=i : i * x for i in range(4)] or from functools import partial from operator import mul def multipliers(): return [partial(mul, i) for i in range(4)]

What will be the output of the code below?
 	
class Parent(object):
x = 1
class Child1(Parent):
pass
class Child2(Parent):
pass
 	
print(Parent.x, Child1.x, Child2.x)
Child1.x = 2
print(Parent.x, Child1.x, Child2.x)
Parent.x = 3
print(Parent.x, Child1.x, Child2.x)
	Answer:
	
	1 1 1
	1 2 1
	3 2 3
	
	Explanation:
	
	in Python, class variables are internally handled as dictionaries. If a variable name is not found in the dictionary of the current class,
	the class hierarchy (i.e., its parent classes) are searched until the referenced variable name is found
	(if the referenced variable name is not found in the class itself or anywhere in its hierarchy, an AttributeError occurs).
	Therefore, setting x = 1 in the Parent class makes the class variable x (with a value of 1) referenceable in that class and any of its children.
	That's why the first print statement outputs 1 1 1.
	Subsequently, if any of its child classes overrides that value (for example, when we execute the statement Child1.x = 2),
	then the value is changed in that child only. That's why the second print statement outputs 1 2 1.
	Finally, if the value is then changed in the Parent (for example, when we execute the statement Parent.x = 3),

What will be the output of the code below in Python 2?
 	
def div1(x,y):
print "%s/%s = %s" % (x, y, x/y)
 	
def div2(x,y):
print "%s//%s = %s" % (x, y, x//y)
 	
div1(5,2)
div1(5.,2)
div2(5,2)
div2(5.,2.)
 	
Also, how would the answer differ in Python 3 (assuming, of course, that the above print statements were converted to Python 3 syntax)?
	Answer: 5/2 = 2
	5.0/2 = 2.5
	5//2 = 2
	5.0//2.0 = 2.0
	
	Explanation: By default, Python 2 automatically performs integer arithmetic if both operands are integers.
	As a result, 5/2 yields 2, while 5./2 yields 2.5. Also note that the "double-slash" (//) operator will always perform integer division,
	regardless of the operand types. That's why 5.0//2.0 yields 2.0 even in Python 2.
	
	(Py 3+)
	
	5/2 = 2.5
	5.0/2 = 2.5
	5//2 = 2
	5.0//2.0 = 2.0

What will be the output of the code below? list = ['a', 'b', 'c', 'd', 'e'] print(list[10:])	Answer: [] Explanation: Attempting to access a slice of a list at a starting index that exceeds the number of members in the list will not result in an IndexError and will simply return an empty list. What makes this a particularly nasty gotcha is that it can lead to bugs that are really hard to track down since no error is raised at runtime.
What will be the ouput of lines 2, 4, 6, and 8? 1. list = [[ ]] * 5 2. list # output? 3. list[0].append(10) 4. list # output? 5. list[1].append(20) 6. list # output? 7. list.append(30) 8. list # output?	Answer: [[], [], [], [], []] [[10], [10], [10], [10], [10]] [[10, 20], [10, 20], [10, 20], [10, 20], [10, 20]] [[10, 20], [10, 20], [10, 20], [10, 20], [10, 20], 30] Explanation: The first line of output is presumably intuitive and easy to understand; i.e., list = [[ ]] * 5 simply creates a list of 5 lists. However, the key thing to understand here is that the statement list = [[ ]] * 5 does NOT create a list containing 5 distinct lists; rather, it creates a a list of 5 references to the same list. With this understanding, we can better understand the rest of the output. list[0].append(10)
Given a list of N numbers, use a single list comprehension to produce a new list that only contains those values that are: (a) even numbers, and (b) from elements in the original list that had even indices For example, if list[2] contains a value that is even, that value should be included in the new list, since it is also at an even index (i.e., 2) in the	Answer: [x for x in list[::2] if x%2 == 0]

Given the following subclass of dictionary:
 	
class DefaultDict(dict):
def __missing__(self, key):
return []
 	
Will the code below work? Why or why not?
 	
d = DefaultDict()
d['florp'] = 127
	Answer: Yes, it will work. With this implementation of the DefaultDict class, whenever a key is missing,
	the instance of the dictionary will automatically be instantiated with a list.

What is Python really? You can (and are encouraged) make comparisons to other technologies in your answer	Answer: Here are a few key points: - Python is an interpreted language. That means that, unlike languages like C and its variants, Python does not need to be compiled before it is run. Other interpreted languages include PHP and Ruby. Python is dynamically typed, this means that you don't need to state the types of variables when you

Fill in the missing code:
 	
def print_directory_contents(sPath):
"""
This function takes the name of a directory
and prints out the paths files within that
directory as well as any files contained in
contained directories.
 	
This function is similar to os.walk. Please don't
use os.walk in your answer. We are interested inyour
ability to work with nested structures.
"""
fill_this_in
	Answer:
	
	def print_directory_contents(sPath):
	import os
	for sChild in os.listdir(sPath):
	sChildPath = os.path.join(sPath,sChild)
	if os.path.isdir(sChildPath):
	print_directory_contents(sChildPath)
	else:
	print(sChildPath)

Looking at the below code, write down the final values of A0, A1, ...An.
 	
A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))
A1 = range(10)
A2 = sorted([i for i in A1 if i in A0])
A3 = sorted([A0[s] for s in A0])
A4 = [i for i in A1 if i in A3]
A5 = {i:i*i for i in A1}
A6 = [[i,i*i] for i in A1]
	Answer:
	
	A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4} # the order may vary
	A1 = range(0, 10) # or [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] in python 2
	A2 = []
	A3 = [1, 3, 2, 5, 4]
	A4 = [1, 2, 3, 4, 5]
	A5 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
	A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64],

Python and multi-threading. Is it a good idea? List some ways to get some Python code to run in a parallel way.	Answer: Python doesn't allow multi-threading in the truest sense of the word. It has a multi-threading package but if you want to multi-thread to speed your code up, then it's usually not a good idea to use it. Python has a construct called the Global Interpreter Lock (GIL). The GIL makes sure that only one of your 'threads' can execute at any one time. A thread acquires the GIL, does a little work, then passes the GIL onto the next thread.

What does this code output:
 	
def f(x,l=[]):
for i in range(x):
l.append(i*i)
print(l)
 	
f(2)
f(3,[3,2,1])
f(3)
	Answer:
	
	[0, 1]
	[3, 2, 1, 0, 1, 4]
	[0, 1, 0, 1, 4]

What is monkey patching and is it ever a good idea?	Answer: Monkey patching is changing the behaviour of a function or object after it has already been defined. For example: import datetime datetime.datetime.now=lambda:datetime.datetime(2012, 12, 12) Most of the time it's a pretty terrible idea - it is usually best if things act in a well-defined way. One reason to monkey patch would be in testing. The mock package is very useful to this end.
What does this stuff mean: *args, **kwargs? And why would we use it? "	Answer: Use *args when we aren't sure how many arguments are going to be passed to a function, or if we want to pass a stored list or tuple of arguments to a function. **kwargs is used when we dont know how many keyword arguments will be passed to a function, or it can be used to pass the values of a dictionary as keyword arguments. The identifiers args and kwargs are a convention, you could also use *bob and **billy but that would not be wise. Here is a little illustration: def f(*args,**kwargs): print(args, kwargs) l = [1,2,3] t = (4,5,6) d = {'a':7,'b':8,'c':9} f() f(1,2,3) # (1, 2, 3) {} f(1,2,3,""groovy"") # (1, 2, 3, 'groovy') {} f(a=1,b=2,c=3) # () {'a': 1, 'c': 3, 'b': 2} f(a=1,b=2,c=3,zzz=""hi"") # () {'a': 1, 'c': 3, 'b': 2, 'zzz': 'hi'} f(1,2,3,a=1,b=2,c=3) # (1, 2, 3) {'a': 1, 'c': 3, 'b': 2} f(*l,**d) # (1, 2, 3) {'a': 7, 'c': 9, 'b': 8} f(*t,**d) # (4, 5, 6) {'a': 7, 'c': 9, 'b': 8} f(1,2,*t) # (1, 2, 4, 5, 6) {} f(q=""winning"",**d) # () {'a': 7, 'q': 'winning', 'c': 9, 'b': 8} f(1,2,*t,q=""winning"",**d) # (1, 2, 4, 5, 6) {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}
What do these mean to you: @classmethod, @staticmethod, @property?	Answer: These are decorators. A decorator is a special kind of function that either takes a function and returns a function, or takes a class and returns a class. The @ symbol is just syntactic sugar that allows you to decorate something in a way that's easy to read. @my_decorator def my_func(stuff): do_things Is equivalent to def my_func(stuff): do_things my_func = my_decorator(my_func)

Consider the following code, what will it output?

```
class A(object):
    def go(self):
        print("go A go!")
    def stop(self):
        print("stop A stop!")
    def pause(self):
        raise Exception("Not Implemented")


class B(A):
    def go(self):
        super(B, self).go()
        print("go B go!")


class C(A):
    def go(self):
        super(C, self).go()
        print("go C go!")
    def stop(self):
        super(C, self).stop()
        print("stop C stop!")


class D(B,C):
    def go(self):
        super(D, self).go()
        print("go D go!")
    def stop(self):
        super(D, self).stop()
        print("stop D stop!")
    def pause(self):
        print("wait D wait!")

class E(B,C): pass

a = A()
b = B()
c = C()
d = D()
e = E()

# specify output from here onwards

a.go()
b.go()
c.go()
d.go()
e.go()

a.stop()
b.stop()
c.stop()
d.stop()
e.stop()

a.pause()
b.pause()
c.pause()
d.pause()
e.pause()
```

Answer:

```
a.go()
# go A go!


b.go()
# go A go!
# go B go!


c.go()
# go A go!
# go C go!


d.go()
# go A go!
# go C go!
# go B go!
# go D go!


e.go()
# go A go!
# go C go!
# go B go!


a.stop()
# stop A stop!


b.stop()
# stop A stop!


c.stop()
# stop A stop!
# stop C stop!


d.stop()
# stop A stop!
# stop C stop!
# stop D stop!


e.stop()
# stop A stop!


a.pause()
# ... Exception: Not Implemented


b.pause()
# ... Exception: Not Implemented


c.pause()
# ... Exception: Not Implemented


d.pause()
# wait D wait!


e.pause()
# ...Exception: Not Implemented
```

Consider the following code, what will it output?

```
class Node(object):
    def __init__(self,sName):
        self._lChildren = []
        self.sName = sName
    def __repr__(self):
        return "Node '{}'".format(self.sName)
    def append(self,*args,**kwargs):
        self._lChildren.append(*args,**kwargs)
    def print_all_1(self):
        print(self)
        for oChild in self._lChildren:
            oChild.print_all_1()
    def print_all_2(self):
        def gen(o):
            lAll = [o,]
            while lAll:
                oNext = lAll.pop(0)
                lAll.extend(oNext._lChildren)
                yield oNext
        for oNode in gen(self):
            print(oNode)

oRoot = Node("root")
oChild1 = Node("child1")
oChild2 = Node("child2")
oChild3 = Node("child3")
oChild4 = Node("child4")
oChild5 = Node("child5")
oChild6 = Node("child6")
oChild7 = Node("child7")
oChild8 = Node("child8")
oChild9 = Node("child9")
oChild10 = Node("child10")

oRoot.append(oChild1)
oRoot.append(oChild2)
oRoot.append(oChild3)
oChild1.append(oChild4)
oChild1.append(oChild5)
oChild2.append(oChild6)
oChild4.append(oChild7)
oChild3.append(oChild8)
oChild3.append(oChild9)
oChild6.append(oChild10)

# specify output from here onwards
oRoot.print_all_1()
oRoot.print_all_2()
```

Answer:

oRoot.print_all_1() prints:

```
Node 'root'
Node 'child1'
Node 'child4'
Node 'child7'
Node 'child5'
Node 'child2'
Node 'child6'
Node 'child10'
Node 'child3'
Node 'child8'
Node 'child9'
```

oRoot.print_all_2() prints:

```
Node 'root'
Node 'child1'
Node 'child2'
Node 'child3'
Node 'child4'
Node 'child5'
Node 'child6'
Node 'child8'
Node 'child9'
Node 'child7'
Node 'child10'
```

Describe Python's garbage collection mechanism in brief.

Answer:

Python maintains a count of the number of references to each object in memory.
If a reference count goes to zero then the associated object is no longer live and the memory allocated to that object can be freed up for something else
occasionally things called ""reference cycles"" happen. The garbage collector periodically looks for these and cleans them up.
An example would be if you have two objects o1 and o2 such that o1.x == o2 and o2.x == o1. If o1 and o2 are not referenced by anything else then they shouldn't be live. But each of them has a reference count

Place the following functions below in order of their efficiency.
They all take in a list of numbers between 0 and 1. The list can be quite long.
An example input list would be [random.random() for i in range(100000)].
How would you prove that your answer is correct?
 	
def f1(lIn):
  l1 = sorted(lIn)
  l2 = [i for i in l1 if i0.5]
  return [i*i for i in l2]
 	
def f2(lIn):
  l1 = [i for i in lIn if i0.5]
  l2 = sorted(l1)
  return [i*i for i in l2]
 	
def f3(lIn):
  l1 = [i*i for i in lIn]
  l2 = sorted(l1)
  return [i for i in l1 if i(0.5*0.5)]
	Answer:
	
	Most to least efficient: f2, f1, f3. To prove that this is the case, you would want to profile your code.
	Python has a lovely profiling package that should do the trick.
	
	import cProfile
	lIn = [random.random() for i in range(100000)]
	cProfile.run('f1(lIn)')
	cProfile.run('f2(lIn)')
	cProfile.run('f3(lIn)')

We have the following code with unknown function f(). In f(), we do not want to use return, instead, we may want to use generator.
 	
for x in f(5):
  print(x)
 	
output:
0 1 8 27 64
	Answer:
	
	def f(n):
	  for x in range(n):
	  yield x**3

What is __init__.py?	Answer: It is used to import a module in a directory, which is called package import. If we have a module, dir1/dir2/mod.py, we put __init__.py in each directories so that we can import the mod like this: import dir1.dir2.mod The __init__.py is usually an empty py file. The hierarchy gives us a convenient way of organizing the files
Build a string with the numbers from 0 to 100, "0123456789101112..."	Answer: ''.join([str(x) for x in range(101)])

Basic file processing: Printing contents of a file.	Answer:
	
	try:
	  with open('filename','r') as f:
	  print(f.read())
	except IOError:
	  print(""No such file exists"")"

What is the full name of shortcut PEP?	Python Enhancement Proposals