CS1010E Final (AY2021/2022, SEM1)

Section 1 Syntax and Python usage

MCQs marks = 2 x 30 = 60 marks, (e.t.=1 hour)

QN	Questions	Answer
	You can consider the code in each question is in a separate file. What will be the output when we run the file in IDLE?	
1	`print(--10)`	*a) 10 b) 9 c) -10 d) -9 e) None of the rest
2	`print(7*8%3)`	*a) 2 b) 14 c) 18 d) 21 e) None of the rest
3	`print(3==2!=1!=0)`	*a) False b) True c) 1 d) 0 e) None of the rest
4	`print({1,2,3}&{2,3,4}|{3,4,5})`	*a) {2, 3, 4, 5} b) {1, 2, 3, 4, 5} c) {1, 2, 3, 4} d) {2, 3} e) None of the rest
5	`print(len(((1,2,(3,4),(5,(6,7))))))`	*a) 4 b) 7 c) 6 d) 1 e) None of the rest

6	`La = [1, [2, [3, [4]]]]`
	`Lb = [La[1], La[1][1]]`
	`Lb[0][1] = 9`
	`print(La)`
		*a) [1, [2, 9]]
		b) [1, [2, [9]]]
		c) [1, [2, [3, [4]]]]
		d) IndexError
		e) None of the rest

7	`print(['abc',['def'],'qrs'][2][1][0])`	*a) 'r' b) 'e' c) '' d) 'f' e) None of the rest

8	`La = [1]*2`
	`Lb = [La]*2`
	`Lb = [Lb[:],Lb[::-1]]`
	`Lb[0][1][0] = 9`
	`print(Lb[1][0])`
		*a) [9, 1]
		b) [1, 9]
		c) [1, 1]
		d) [9, 9]
		e) None of the rest

9	`print((1,2,3,[4,5,6],7,8,9)[2:5][::-1])`	*a) [(7, [4, 5, 6], 3) b) ([4, 5, 6], 3, 2) c) (8, 7, [4, 5, 6]) d) (8, 7, [6, 5, 4]) e) None of the rest

10	`def f(x):`
	`  if x % 2 == 1:`
	`  return x+1`
		*a) 10
		b) 12
		c) 14

	else: return x+4 print(f(f(f((1)))))	d) 16 e) None of the rest

11	def foo(x):
	if not x:
	return x
	return x[-1] + foo(x[:-1]) + x[-1]
	print(foo('abcde'))
		*a) 'edcbaabcde'
		b) 'abcdeedcba'
		c) 'abcdedcba'
		d) 'abcdeebcda'
		e) 'edcbabcde'

12	def f(x):
	if x < 0:
	return 'f'
	if x%2==0:
	return g(x-1)
	return f(x-1)
	def g(x):
	if x < 0:
	return 'g'
	if x%2==1:
	return f(x-1)
	return g(x-1)
	print(f(100))
		*a) 'g'
		b) 'f'
		c) The code will run into an infinite loop
		d) It will print a very long string with alternating 'f' and 'g'
		e) It will cause a RecursionError exception

13	x,y = 100,0
	if x > 10:
	y = 1
	elif x > 100:
	y = 2
	if x > 1000:
	y = 3
	else:
	y = 4
	print(y)
		*a) 4
		b) 3
		c) 2
		d) 1
		e) None of the rest

14	x,y = 10,20
	if x > y:
	z = 1
	if 2*x == y:
	z = 2
	else:
	if 2*x < y:
	z = 3
	elif 2*x > y:
	z = 4
	print(z)
		a) 1
		b) 2
		c) 3
		d) 4
		*e) None of the rest
		
		note: the path does not exists

15	Consider the following code:
	`if x > 0:`
	`  x = x+2`
	`if x < 10:`
	`  y = 4`
	`elif x > 20:`
	`  y = 3`
	`else:`
	`  if x < 5:`
	`  y = 2`
	`  else:`
	`  y = 1`
	`print(y)`
	Which of the following output is impossible given any numeric value of x?
		*a) 1
		b) 2
		c) 3
		d) 4
		e) All outputs are possible

16	`x = 'a'`
	`while len(x)<6:`
	`  x += 'bc'`
	`print(x)`
		*a) 'abcbcbc'
		b) 'abcbc'
		c) 'abcbcbcbc'
		d) 'abcbcb'
		e) None of the rest

17	`a = [1,2,3,4]`
	`x = 0`
	`for i in range(len(a)):`
	`  x += 1`
	`  if i in a:`
	`  a.remove(i)`
	`print(x,a)`
		*a) 4 [4]
		b) 2 [3, 4]
		c) 2 [2, 3, 4]
		d) 4 [3, 4]
		e) 4 [2, 3, 4]

18	`x,y = 9999,0`
	`while x > 0:`
	`  y += 6`
	`  x -= 6`
	`print(y)`
		*a) 10002
		b) 10001
		c) 10000
		d) 9999
		e) None of the rest

19	`x = 0`
	`i = 0`
	`while i < 7:`
	`  if i%2 == 0:`
	`  i += 1`
	`  if i%3 == 0:`
	`  continue`
	`  i += 1`
	`  x += i`
	`print(x)`
		a) 19
		b) 12
		c) 25
		d) 21
		*e) None of the rest
		
		note: it's an infinite loop because continue skips i += 1

20	`x = [1,2]`
	`def foo(y):`
	`  y[0] = 9`
	`  x[0] = 3`
	`foo(x)`
	`print(x)`
		*a) [3, 2]
		b) [9, 2]
		c) [1, 2]
		d) [1, 3]
		e) [1, 9]

21	y = 12
	def foo(x):
	  if x < 0:
	  y = 1
	  elif x > 0:
	  y -= 1
	  else:
	  y = x+1
	print(foo(y))
		*a) None of the rest
		b) 1
		c) 13
		d) 12
		e) None

22	print({'a':'d','c':'e','f':'z'}[{1:'f',5:'c',9:'a'}[1]])	*a) 'z' b) 'e' c) 'd' d) '' e) None of the rest

23	def f1(x):
	  return x+1
	def g1(x):
	  return x-1
	d1 = {1:f1,2:g1}
	x = 1
	for i in range(10):
	  x = d1[x](x)
	print(x)
		*a) 1
		b) 2
		c) Run into infinite loop
		d) Raise a RecursionError exception
		e) Raise a KeyError exception

24	def tail(lst,op,res):
	  if not lst:
	  return res
	  else:
	  return tail(lst[:-1],op,op(res,lst[-1]))
	print(tail([1,2,3,4], lambda x,y: (y,x), ()))
		*a) (1, (2, (3, (4, ()))))
		b) (((((), 4), 3), 2), 1)
		c) (4, (3, (2, (1, ()))))
		d) (((((), 1), 2), 3), 4)
		e) None of the rest

25	f = lambda x,y: lambda z: (x)(y)(z) print((f)(lambda x: lambda y: x, lambda z: z*2)(3)(4))	*a) 8 b) 6 c) TypeError d) RecursionError e) None of the rest
26	f1 = lambda x,y: x f2 = lambda x,y: y f3 = lambda c,x,y: c(x,y) print(f3(f1,f2,f3)(f2,f1)(1,0))	*a) 1 b) 0 c) TypeError d) RecursionError e) None of the rest

27	class C0:
	  def __init__(self,n):
	  member = n*n
	  def cout(self):
	  print(member)
	x = C0(5)
	x.cout()
		*a) None of the rest
		b) 5
		c) 25
		d) None
		e) 0

28	class C1():
	  def __init__(self, a):
	  self.x = a
	  def f(self):
	  return self.y
	class C2(C1):
	  def __init__(self, a):
	  self.y = a
	  super().__init__(a+a)
	  def f(self):
	  return super().f() + self.x
	class C3(C2):
	  def __init__(self, a):
	  self.x = a
	  super().__init__(a+a)
	print(C3(2).f())
		*a) 12
		b) 16
		c) 6
		d) 8
		e) None of the rest

29	class C1():
	  def f(self):
	  return 1 + self.g()
	  def g(self):
	  return 2
	class C2(C1):
	  def f(self):
	  return 3 + super().f()
	class C3(C2):
	  def g(self):
	  return 4
	print(C3().f())
		a) 6
		b) 7
		*c) 8
		d) 9
		e) None of the rest

30	output = ''
	try:
	  x = 2
	  y = 'a'
	  q = y + str(x)
	  z = q * x
	except TypeError:
	  output += '1'
	else:
	  output += '2'
	finally:
	  output += '3'
	print(output)
		a) ''
		b) '13'
		c) '3'
		*d) '23'
		e) None of the rest

	You are given three functions and their have the same functionality. Namely, given a list L, they will extract the element in L that is unique. E.g.
	`>>> extract_unique_1([1,2,3,4,4,4,5])`
	`[1, 2, 3, 5]`
	Here are the three versions, they are called versions 1, 2 and 3 respectively according to their function name
	`def extract_unique_1(L):`
	`  output = []`
	`  for i in L:`
	`  if L.count(i) == 1:`
	`  output.append(i)`
	`  return output`
	
	`def extract_unique_2(L):`
	`  L2 = sorted(L)`
	`  output = []`
	`  for i in range(len(L2)):`
	`  unique = True`
	`  if i > 0:`
	`  if L2[i] == L2[i-1]:`
	`  unique = False`
	`  if i < len(L2)-1:`
	`  if L2[i] == L2[i+1]:`
	`  unique = False`
	`  if unique:`
	`  output.append(L2[i])`
	`  return output`
	
	`def extract_unique_3(L):`
	`  d = dict()`
	`  for i in L:`
	`  if not i in d:`
	`  d[i] = 0`
	`  d[i] += 1`
	`  output = list(filter(lambda x:d[x]==1,L))`
	`  return output`
	Answer the following three questions assuming that the input is a very long list, e.g. `len(L)>10000`.	

31	Which version(s) is/are significantly slowest?	*a) 1 b) 2 c) 3 d) 1 and 2 e) 2 and 3 f) 1 and 3 g) They are all equally slow

32	Which version(s) is/are significantly fastest?	a) 1 *b) 2 *c) 3 d) 1 and 2 *e) 2 and 3 f) 1 and 3 g) They are all equally fast
33	If we really want to choose one version that is the fastest, which one will it be?	a) 1 b) 2 *c) 3 d) It is impossible to determine which one is the fastest. e) They are all the same speed

34	Oh no, the following code does not work!
	
	`def swap(lst, i, j):`
	`  lst[i],lst[j] = lst[j],lst[i]`
	`  return lst`
	`def reverse(lst):`
	`  n = len(lst)`
	`  for i in range(n):`
	`  n -= 1`
	`  swap(lst,i,n)`
	`  return lst`
	
	The function is supposed to reverse the list but when we try with `reverse([1,2,3,4,5,6])`, the result is still `[1,2,3,4,5,6]`.
	
	Which changes should be made so that the function can reverse the input list `lst` correctly?
		*a) `range(n//2)` instead of "`range(n)`" at line 6
		b) `lst = swap(lst,i,n-i-1)` instead of "`swap(lst,i,n-i-1)`" at line 8
		c) `n = len(lst)//2` instead of "`n = len(lst)`" at line 5
		d) `swap(lst,i,n-1)` instead of "`swap(lst,i,n)`" at line 8 and remove "`n -= 1`" at line 7
		e) None of the rest changes works

35	What is the output for the following code?
	`def foo(x):`
	`  if not x:`
	`  return x`
	`  if type(x) != tuple:`
	`  return (x,)`
	`  return foo(x[1:]) + foo(x[0])`
	`print(foo((0,1,2,3,(4,(5,6)))))`
		*a) None of the rest
		b) (6,5,4,3,2,1,0)
		c) (((6,5),4),3,2,1,0)
		d) (((5,6),4),3,2,1,0)
		e) ((4,(5,6)),3,2,1,0)

36	Which of the following function call returns False?
	`def foo(x):`
	`  if len(x) <= 1:`
	`  return True`
	`  else:`
	`  m = len(x)//2`
	`  return foo(x[m]) and x[0] == x[-1]`
		*a) "[0,[1,[2,[],2],1],0]"
		b) [0,[1,[2,[],2],1],0]
		c) "abcda"
		d) [0,[1,[2,3],1],0]
		e) None of the rest

37	What is your comment for the following code to ask a user to choose between 'coffee' or 'tea'?
	`try:`
	`  print("Do you want coffee or tea?")`
	`  print("If you want coffee, please enter \'c\'")`
	`  print("Any other choices will be deemed as tea")`
	`  ans = input()`
	`  assert ans == 'c'`
	`except:`
	`  request = 'Tea'`
	`else:`
	`  request = 'Coffee'`
	`finally:`
	`  print("Enjoy your "+request+"!")`
		*a) The code is abusing the usage of exceptions
		b) The code is too inefficient
		c) The code may give the wrong answer for the choice from the user
		d) The code may crash
		e) The code is perfect and it is implemented in the best way.

38	In the memoization Fibonacci example mentioned in our lecture slides, what data structure did we use to improve the efficiency of the code to the best?	*a) dictionary b) set c) list d) tuple e) lambda function
39	What is the best and simplest method to read in a .csv file into a 2D array?	*a) Use the csv package b) Use the built-in file opening functions of Python c) Use the imagio package d) Use the PIL package e) None of the rest

40	What is the best data type to store a number that is bigger than 2**1024?	*a) integer b) float c) string d) lambda function e) None of the rest

	For all the fill-in-the-blank questions, please do not enter spaces before (on the left side) of your answers. Or it will be deemed as wrong answer	

41	You are given a 2D map like the following:
	`map1 = ['..........',`
	`'......(+)..',`
	`'..........',`
	`'....(+)....']`
	And you want to locate all the positions of the star '(+)'. E.g.
	`>>> find_star(map1)`
	`(1, 7)`
	`(3, 5)`
	Fill in the blanks in the following code to complete the function `find_star()`:
	`def find_star(m):`
	`n_row = len(m)`
	`n_col = len(m[0])`
	`for i in range(__1__):`
	`for j in range(__2__):`
	`if __3__:`
	`print((i,j))`
		`def find_star(m):`
		`n_row = len(m)`
		`n_col = len(m[0])`
		`for i in range(n_row):`
		`for j in range(1,n_col-1):`
		`if m[i][j] == '+' and m[i][j-1] == '(' and m[i][j+1] == ')':`
		
		`print((i,j))`

42	Here is the same partial code from your OOP assignments. class Fighter(Character):
	`def __init__(self):`
	`super().__init__()`
	`self.name = 'Fighter'`
	`self.maxhp = 1200`
	`self.hp = 1200`
	`self.str = 100`
	`self.cost = 100`
	
	`def act(self,myTeam,enemy):`
	`target = randAlive(enemy)`
	`dprint(f'Hurt enemy {target} by damage {self.str}.')`
	
	`enemy[target].gotHurt(self.str)`
	Fill in the blanks in the following to implement the class SickFighter. The SickFighter will have the same maxhp as the Fighter but his hp is just half of his maxhp at the beginning. And he will act in the same way the Figther does except that his hp will be reduced by 1 whenever he acts. However, his action will not cause his hp to drop to zero. Meaning, if he acts and his hp is 1, his hp will remains at 1. (You don't need to change the 'name' attribute of the SickFighter)
	
	`class SickFighter(Fighter):`
	`def __init__(self):`
	`__1__`
	`__2__`
	`def act(self,myTeam,enemy):`
	`__3__`
	`__4__`
	`__5__`
	
	Please be reminded that you do not need to put any spaces before your answers. You can following the indentations given in the skeleton code
		`class SickFighter(Fighter):`
		`def __init__(self):`
		
		`super().__init__()`
		`self.hp /= 2`
		`def act(self,myTeam,enemy):`
		`if self.hp>1:`
		`self.hp-=1`
		
		`super().act(myTeam,enemy)`

<table>
<tr>
<td>43</td>
<td>The greatest common divisor (gcd) of two numbers n and m is a number x such that x fully divides both n and m. At the worst case, the gcd is 1.
We want to write an iterative function to find the gcd of two positive numbers.

For example, <code>gcd(20,10)</code> should return 10 and <code>gcd(5,7)</code> should return 1.

You are given the following code. Fill in the blank to complete the code.

<pre>def gcd(n,m):
  res = 0
  for i in __1__:
    if __2__:
      __3__
    return res</pre></td>
<td>Solution #1
<pre>def gcd(n,m):
  res = 0
  for i in
range(1,min(n,m)+1):
    if n%i == m%i == 0:
      res = i
  return res</pre>
Solution #2
<pre>def gcd(n,m):
  res = 0
  for i in
range(min(n,m),0,-1):
    if n%i == m%i == 0:
      return i
  return res</pre></td>
</tr>
<tr>
<td>44</td>
<td>A prime divisor of n is a prime number that can fully divide a number n. For instance, the number 12 has 3 (non-unique) prime divisors: 2, 2 and 3.

We want to count the number of non-unique prime divisors of n by using a recursive helper function. Note that by continuously dividing a number by its smallest divisors, the next number that can divide it is guaranteed to be a prime number.

Complete the code below to count the non-unique prime divisors of n. You may assume that n > 1. For example, <code>count_prime_div(12)</code> should return 3 and <code>count_prime_div(11)</code> should return 1.

<pre>def count_prime_div(n):
  def helper(n,div):
    if div > n:
      return 0
    if __1__:
      return 1
    if n%div == 0:
      return __2__
    else:
      return __3__
    return helper(n,2) # first prime is
  2</pre></td>
<td><pre>def count_prime_div(n):
  def helper(n,div):
    if div > n:
      return 0
    if div == n:
      return 1
    if n%div == 0:
      return 1 +
helper(n//div,div)
    else:
      return
helper(n,div+1)
  return helper(n,2) #
first prime is 2</pre></td>
</tr>
</table>