1) Execute the following snippet of code and then answer questions 1 and 2.

```
import random

rolls = [ ]
for i in range(100000):
    # randint includes the endpoints
    roll = random.randint(1, 6)
    rolls.append(roll)

count = 0
primes = [2, 3, 5]
for roll in rolls:
    if roll in primes:
        count += 1
```

What is the size of the list `rolls`? (NAT)

100000

2 points

2) Execute the following snippet of code. ***2 points***

```
some_var = count / len(rolls)
```

What does the variable `some_var` represent?

- It represents the probability that a number chosen at random between 1 and 100000 is a prime.
- It represents the number of primes between 1 and 100000
- It represents the number of prime numbers in the list `rolls`.
- It represents the probability that a number chosen at random from the list `rolls` is a prime.
- It represents the probability that a number chosen at random from the list `primes` is a prime.

3) **Common Data for questions (3) and (4)** ***2 points***

Assume that `L` is a non-empty list of positive integers. Also assume that the list is a distinct collection of numbers, i.e., no two numbers are alike. Consider the following code. Answer questions 3 and 4 based on this code.

```
S = 0
for x in L:
    S += x

flag = False
y = -1
for x in L:
    if x * len(L) == S:
        flag = True
        y = x
        break
```

If `flag` is `True` at the end of execution of the code given above, which of the following statements are true? Note that the options should be true for any list `L` that satisfies the conditions given in the common data. Multiple options could be correct.

- `y` is an element in the list `L`

- [ ] `y` is the smallest number in the list
- [ ] `y` is the greatest number in the list
- [x] `y` is the average (arithmetic mean) of the numbers in the list
- [ ] `y` is the element at index `len(L) // 2` in the list `L`

4) Assume that `L` is a list of the first n positive integers, where $n > 0$. Under what conditions will the variable `flag` be `True` at the end of execution of the code given above? ***2 points***

- (•) `n` is an odd integer
- ( ) `n` is an even integer

5) What is the output of the following snippet of code? ***2 points***

Hint:

(1) If `L = [1, 2, 3, 4, 5]`, then `L[1: 3]` is the list `[2, 3]`. Slicing a list is very similar to slicing a string. All the rules that you have learned about string-slicing apply to list-slicing.

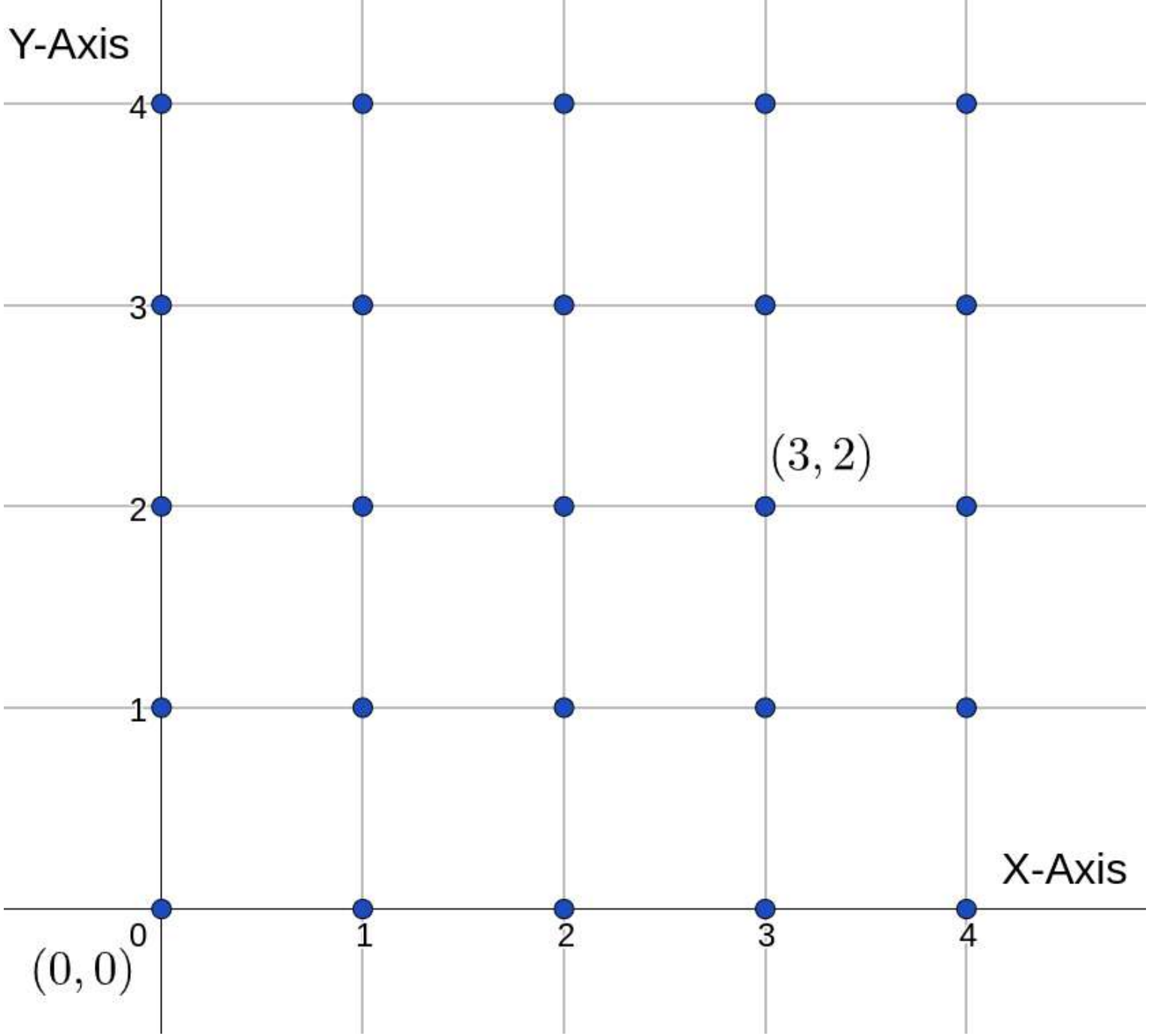
(2) If `P = [1, 2, 3]` and `Q = [4, 5, 6]` then `P + Q` is the list `[1, 2, 3, 4, 5, 6]`. Again, list concatenation is very similar to string concatenation.

```
1 L = [90, 47, 8, 18, 10, 7]
2 S = [L[0]]  # list containing just one element
3 for i in range(1, len(L)):
4     flag = True
5     for j in range(len(S)):
6         if L[i] < S[j]:
7             before_j = S[: j]   # elements in S before index j
8             new_j = [L[i]]      # list containing just one element
```

```
            after_j = S[j: ]    # elements in S starting from index j
            # what is the size of S now?
            S = before_j + new_j + after_j
            # what is the size of S now?
            flag = False
            break
    if flag:
        S.append(L[i])
print(S)
```

- [ ] `[90, 47, 8, 18, 10, 7]`
- [ ] `[7, 10, 18, 8, 47, 90]`
- [x] `[7, 8, 10, 18, 47, 90]`
- [ ] `[90, 47, 18, 10, 8, 7]`
- [ ] `[90, 7, 8, 10, 18 47]`

6) Consider the following grid of integer points in the plane: $P_{ij} = (i, j),\ 0 \leq i, j \leq 4.$ ***2 points***



We wish to represent this grid of points as a list of tuples. The name of the list should be `points`. Each element of the list should be a tuple of the form `(x, y)`. The first element of the tuple is the x-coordinate (horizontal) of the point, the second is the y-coordinate (vertical).

We have employed the standard Cartesian coordinate system. Select all the correct implementations of this program. (MSQ)

```
points = [ ]
for x in range(0, 5):
    for y in range(0, 5):
        points.append(x, y)
```

```
points = [ ]
for x in range(0, 5):
    for y in range(0, 5):
        points.append([x, y])
```

```
points = ( )
for x in range(0, 5):
    for y in range(0, 5):
        points.append((x, y))
```

```
points = [ ]
for x in range(0, 5):
    for y in range(0, 5):
        points.append((x, y))
```

7)

```
L = [y - x for x in [1, 2, 3] for y in [3, 4, 5] if y > x]
```

2 points

Which of the following codes are equivalent to the above code? [MSQ]

```
L = [ ]
```

- [x]
```
for x in [1, 2, 3]:
    for y in [3, 4, 5]:
        if y > x:
            L.append(y - x)
```

- [ ]
```
L = [ ]
for y in [3, 4, 5]:
    for x in [1, 2, 3]:
        if y > x:
            L.append(y - x)
```

- [x]
```
L = [ ]
for x in [1, 2, 3]:
    for y in [3, 4, 5]:
        if y > x:
            L += [y - x]
```

- [ ]
```
L = []
for y in [3, 4, 5]:
    for x in [1, 2, 3]:
        if y > x:
            L += [y -x]
```

8) We wish to find all integer triplets (x, y, z) such that: ***2 points***

$x^2 + y^2 = z^2$, and $0 < x < y < z < 100$

Select all snippets of code that create a list called `triplets` and store each triplet as a tuple. [MSQ]

- [x]
```
triplets = [(x, y, z) for x in range(1, 100)
                      for y in range(x + 1, 100)
                      for z in range(y + 1, 100)
                      if x ** 2 + y ** 2 == z ** 2]
```

- [x]
```
triplets = [ ]
for x in range(1, 100):
    for y in range(x + 1, 100):
        for z in range(y + 1, 100):
            if x ** 2 + y ** 2 == z ** 2:
                triplets.append((x, y, z))
```

- [x]
```
triplets = [(x, y, z) for x in range(1, 100)
                      for y in range(1, 100)
                      for z in range(1, 100)
                      if x ** 2 + y ** 2 == z ** 2 and x < y < z]
```

9) L is a list of names. Create a list P of names that contain only those names in L that begin with a capital letter. Select all correct implementations. [MSQ] ***2 points***

- [ ]
```
P = [name for name in L if 'a' <= name[0] <= 'z']
```

- [x]
```
P = [name for name in L if 'A' <= name[0] <= 'Z']
```

```
P = [ ]
for name in L:
    if 'A' <= name[0] <= 'Z':
        P.append(name)
```

10) Accept a sequence of comma-separated strings as input from the user and populate a list of strings that do not have the letter 'e' in them. Print this list as output to the console. Select all snippets of code that achieve this. [MSQ] ***2 points***

```
P = input().split(',')
L = [ ]
for word in P:
    if 'e' not in word:
        L.append(word)
print(L)
```

```
L = [word for word in input().split(',') if 'e' not in word]
print(L)
```

```
print([word for word in input().split(',') if 'e' not in word])
```

11) Consider the following snippet of code. ***2 points***

```
def minmax(a, b):
    if a <= b:
```

```
3         return a, b
4     return b, a
```

x is a real number. When `minmax(x, x)` is called, which `return` statement in the function is executed?

- The return statement in line-3 which is inside the `if-block`.
- The return statement in line-4 which is outside the `if-block`.
- Both the return statements are executed.
- Neither return statement is executed.

12) Select all correct implementations of a function named `unique` that accepts a non-empty list `L` of integers as an argument. The function should remove all duplicate elements from the list `L`. Specifically, It should return a list `L_uniq` that retains the *first occurrence* (from the left) of each distinct element in the input list `L`. ***2 points***

A few instances of the input-output behaviour of the function is given below. Note that these are just sample test cases. Your function should work for any non-empty list `L` of integers. (MSQ)

L	unique(L)
[1, 1, 2, 3, 5, 5]	[1, 2, 3, 5]
[1, 2, 3, 4, 5, 7, 7]	[1, 2, 3, 4, 5, 7]

- [x]
```
def unique(L):
    L_uniq = [ ]
    for elem in L:
        if elem not in L_uniq:
            L_uniq.append(elem)
```

```
    return L_uniq
```

```
def unique(L):
    L_uniq = [ ]
    for elem in L:
        if elem in L_uniq:
            L_uniq.append(elem)
    return L_uniq
```

```
def unique(L):
    L_uniq = [L[0]]
    for i in range(1, len(L)):
        if not(L[i] in L[:i]):
            L_uniq.append(L[i])
    return L_uniq
```

```
def unique(L):
    L_uniq = [ ]
    for i in range(1, len(L)):
        if not(L[i] in L[:i]):
            L_uniq.append(L[i])
    return L_uniq
```

```
def unique(L):
    L_uniq = [ ]
    for i in range(0, len(L)):
        if not(L[i] in L[i + 1: ]):
            L_uniq.append(L[i])
```

```
    return L_uniq
```

13) Given a Python list of the coefficients of a polynomial – `L = [a_0, a_1, a_2,. . ., a_n]` – write a function `poly` that accepts the list of coefficients `L` and a real number `x_0` as arguments. It should return the polynomial evaluated at the value `x_0`. ***2 points***

For example `poly([1, 2, 3], 5)` should return the value $1 + 2 \times 5 + 3 \times 5^2 = 86$. Select the correct implementation of this function.

```
def poly(L, x_0):
    n = len(L)
    for i in range(n):
        psum = psum + L[i] * (x_0 ** i)
    return psum
```

```
def poly(L, x_0):
    psum = 0
    n = len(L)
    for i in range(1, n):
        psum = psum + L[i] * (x_0 ** i)
    return psum
```

```
def poly(L, x_0):
    psum = 0
    n = len(L)
    for i in range(n):
        psum = psum + L[i] * (x_0 ** i)
    return psum
```

```
def poly(L, x_0):
    psum = 0
    n = len(L)
    for i in range(n):
        psum = psum + x_0 * (L[i] ** i)
    return psum
```

14) Write a function named `poly_zeros` that accepts the list of coefficients `L`, and two integers `a` and `b` as arguments. It should return a list named `zeros`, that consists of all integer-zeros of the polynomial in the range $[a, b]$, endpoints inclusive, in ascending order. For example, `poly_zeros([2, -3, 1], 0, 4)` should return the list `[1, 2]`. ***2 points***

Assume that you have access to the function `poly` that was defined in question (7). Select the correct implementation of this function.

Note: The returned list should not have any repeating elements.

```
def poly_zeros(L, a, b):
    zeros = [ ]
    for x in range(a, b + 1):
        if poly(L, x) != 0:
            zeros.append(x)
    return zeros
```

```
def poly_zeros(L, a, b):
    zeros = [ ]
    for x in range(a, b + 1):
        if poly(L, x) == 0:
            zeros.append(x)
        else:
            return zeros
```

```
    return zeros
```

```
def poly_zeros(L, a, b):
    zeros = [ ]
    for x in range(a, b + 1):
        if poly(L, x) == 0:
            zeros.append(x)
```

```
def poly_zeros(L, a, b):
    zeros = [ ]
    for x in range(a, b + 1):
        if poly(L, x) == 0:
            zeros.append(x)
    return zeros
```

15) Using the two functions defined above, find the number of integer-zeros of the polynomial $f(x) = x^6 - 4x^5 - 18x^4 + 52x^3 + 101x^2 - 144x - 180$ in the range $[0, 4]$, endpoints inclusive. You must enter a non-negative integer. Note that we are asking for the number of integer-zeros and not their values. (NAT)

2

2 points