# CS101 midterm exam (2)

- Be sure to enter your information to start
- Do not turn this page until instructed to
- There are 10 multi-choice questions and 3 programming problems to finish
- You must not communicate with other students during the exam
- You must not use any electronic devices during the exam
- Any violations detected will result in a penalty up to full credit loss
- This is a 50-minute exam

## **Part A**, Fill in your information

| | |
|---|---|
| *Full Name* | : ________________________ |
| *UIN (student ID)* | : ________________________ |
| *Session (A/B)* | : ________________________ |

## **Part B**, multi-choice questions

---

*This part has 10 questions for python programming language basics. There is only one correct answer to each question unless otherwise stated. <u>Please fill in your answers on the back of page 2.</u>*

1. (1 point) Consider the following program:

```
import numpy as np
x = np.zeros((3,3))
for i in range(3):
    for j in range(3):
        x[i, j] = i==j
```

(A) $\begin{bmatrix} 0. & 0. & 0. \\ 0. & 1. & 0. \\ 0. & 0. & 1. \end{bmatrix}$

(B) $\begin{bmatrix} 0 & 0. & 0. \\ 0. & 1 & 0. \\ 0. & 0. & 2 \end{bmatrix}$

(C) $\begin{bmatrix} 1. & 0. & 0. \\ 0. & 1. & 0. \\ 0. & 0. & 1. \end{bmatrix}$

(D) $\begin{bmatrix} True & 0. & 0. \\ 0. & True & 0. \\ 0. & 0. & True \end{bmatrix}$

2. (1 point) Consider the following incomplete function

```
def pal(s):
    a = list(s)
    n = len(a)
    ???
```

This function is intended to return **True** if and only if the input string s is a palindrome. A palindrome is a string that reads the same forward and backward, like ‘ABBA’ or ‘EYE’. Which of the following does NOT complete the function definition correctly?

(A)
```
b = a[:]
a.reverse()
return a == b
```

(B)
```
for i in range(n):
    if a[i] != a[n-i-1]:
        return False
return True
```

(C) `return a[: :] == a[n-1:0:-1]`

(D) None of the above

3. (1 point) Which of the following does not produce the behavior of tossing a coin 10 times?

(A)
```
x = np.array([0, 1])
d = np.random.choice(x, 10)
```

(B) `d = np.random.randint(2, size=(10,))`

(C) `x = np.random.uniform(0,2, size=(10,))`

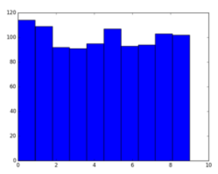
(D) `d = np.random.uniform(size=(10,)) < 0.5`

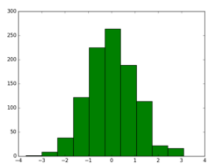
(E) None of the above

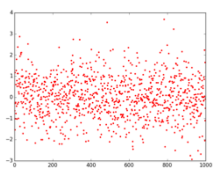
4. (1 point) Consider the following code:

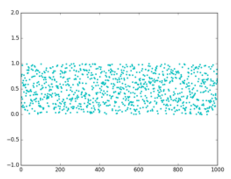
```
x = np.random.randn(size=(1000,1))
plt.hist(x)
plt.show()
```

Which is a possible output of this code?


(A)


(B)


(C)


(D)

5. (1 point) Which of the following code produces plot (D) in Problem 4.

(A) `plt.hist(np.random.randint(1000, size=(1000, 1))); plt.show()`

(B) `plt.plot(np.random.randint(1000, size=(1000, 1))) ; plt.show()`

(C) `plt.plot(np.random.uniform(size=(1000, 1))) ; plt.show()`

(D) `plt.plot(np.random.randn(size=(1000, 1))) ; plt.show()`

6. (1 point) Given the following 2D array :

$$d = \begin{bmatrix} 5 & 2 & 1 & 4 \\ 3 & 1 & 2 & 1 \\ 3 & 4 & 1 & 8 \end{bmatrix}$$

Which of the following produces the subarray np.array([[1,2],[4,1]])?

(A) d[0:2][2:4]

(B) d[0:2,2:4]

(C) d[1:3][1:3]

(D) d[1:3][:,1:3]

(E) None of the above

7. (1 point) Consider the following program:

```
x = np.array([1, 2] * 2 + [1 1]) + 1
```

What is the final value of x?

(A) [1 2 1 2 1 1 1]

(B) [2 3 2 3 2 2]

(C) [4 6]

(D) $\begin{bmatrix} 3 & 6 \\ 2 & 2 \end{bmatrix}$

(E) None of the above

8. (1 point) Consider the following code:

```
a = list(range(6))
count = 0
for i in range(1,len(a)):
        for j in itertools.combinations(a, i):
                count += 1
```

What is the final value of *count*?

(A) 64

(B) 63

(C) 31

(D) 32

(E) None of the above

9. (1 point) Given data of economic growth in the past 5 years, which of the following should be taken as the most probable model that explains the data?

(A) A complex model that does not fit the data

(B) A complex model that exactly fits the data

(C) A simpler model that closely fits the data

(D) A simpler model that loosely fits the data

10. (1 point) Which of the following are common strategies in numerical optimization?
(A) divide-and-conquer
(B) randomization
(C) random walk
(D) hill-climbing
(E) steepest ascent
(F) brute force search
(There are possibly multiple correct answers. Try pick them all)

**Your Final Answers for Part B here**:

| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|
| C | C | C | B | C | D | B | E | C | BCDEF |

## **Part C**, programming hackathon

-----------------------------------------------------------------------------------------------------------------------------

*This part has 3 programming problems to finish. You shall only use python as the programming language. There may be different ways to write the programs. You may take whichever way that achieves the goal. Please read the problem definitions carefully.*

11. (3 points) **Dictionary and Mapping**

Given a dictionary type variable ***zip2city*** and a dictionary type variable ***person2city***. *person2city* has person's name as key and name of the city the person lives as value, and zip2city has zip code as key and city name as value. Assume each city uses one zip code. Write a function definition ***getzip*** that takes a person's name as argument, and returns the zip code of the city he/she lives.

```
def getzip(name)
    for zipcode in zip2city:
        if zip2city[zipcode] == person2city[name]
            return zipcode
    return None
```

13. **Numerical Optimization**

Given a function **f**(x): *x* is a 3-dimensional array, each dimension is a float number in the range *[0, 1)*, and the return value of **f** is also a floating point number. You have access to function f (that is, you can call f(x) and get its returned value), but have no information of its implementation detail. In the next, you are going to try and find out the optimal value **x*** with which f(x*) has the maximum value.

a) (3 points) write out a "**brute-force searching**" algorithm to find out an approximate x*. Hint: You can subdivide the range [0, 1) into a set of discrete levels 0.00, 0.01, 0.02, …0.99 (use *numpy.arange(0, 1, 0.01)* ) for each dimension, and then enumerate all the possible combinations.

```
import numpy as np
best_x = np.array([0,0,0])
best_val = f(best_x)
for x0 in np.arange(0,1,0.01):
    for x1 in np.arange(0,1,0.01):
        for x2 in np.arange(0,1,0.01):
            x = np.array([x0,x1,x2])
            if f(x) > best_val:
                best_x = x
                best_val = f(best_x)

print(best_x, best_val)
```

```
import itertools
best_x = np.array([0,0,0])
best_val = f(best_x)
xs = np.arange(0,1,0.01)
for p in itertools.product(xs, repeat=3):
    x = np.array(p)
    if f(x) > best_val:
        best_x = x
        best_val = f(best_x)

print(best_x, best_val)
```

b) (3 points) Now, write a stochastic version of the optimization: the idea is to randomly sample values of x in its domain (*numpy.random.uniform(size=(3,))* ), and take the one that yields the largest value. This strategy is called ***random sampling***. You need to decide a reasonable number of samples to try in your program.

```
best_x = np.random.uniform(size(3,))
best_val = f(best_x)
for t in range(100000):
    sample = np.random.uniform(size(3,))
    if f(sample) > best_val
        best_x = sample
        best_val = f(best_x)
print(best_x, best_val)
```

c) (3 points) Now, you should be ready to implement the ***random walk*** version of the optimization. Your program should start with an initial value $x_0$, then simulate a sequence of random walks of x, and return the x that produces the largest value f(x). In each timestep of the random walk, generate a *small* perturbation $x_{trial}$ from the current position $x_{current}$: if the trial solution produces a larger value of f, then accept it; if it produces a smaller value (down-hilling), then accept it with a probability. You shall choose a probability function to accept the down-hilling steps that makes sense, for example, a probability based on the percentage of value decrease.

```
best_x = np.random.uniform(size(3,))
best_val = f(best_x)
cur_x = best_x
cur_val = best_val
for t in range(100000):
    trial_x = cur_x + (np.random.uniform(size(3,))-0.5) * 0.002        #stepsize <= 0.001
    (skip testing the case when it goes out of the range)
    trial_val = f(trial_x)
    if trial_val > cur_val:
        cur_x, cur_val = trial_x, trial_val
        if trial_val > best_val:
            best_x, best_val = trial_x, trial_val
    else:
        if np.random.uniform() < 1-abs((trial_val-cur_val)/cur_val):
            cur_x, cur_val = trial_x, trial_val

print(best_x, best_val)
```

12. **Finding $\pi$**

a) (3 points) Figuring out the value of Pi was a mathematical puzzle for a long time. Now with randomization and computing we can easily get a fairly good estimate of Pi by uniformly sampling in a square and counting the number of samples in a circle. Write the python code for this process.

```
import numpy as np
samples = 10000
count = 0
for i in range(samples):
    x,y = np.random.uniform(), np.random.uniform()
    if (x**2 + y **2) < 1:
        count += 1
PI = count/samples*4
```

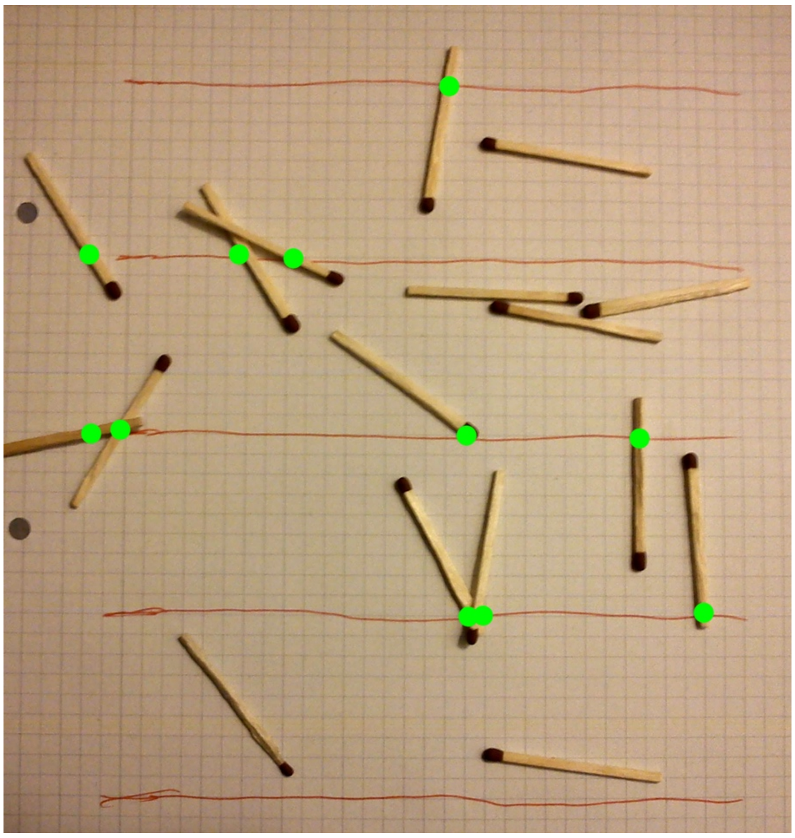
b) (**BONUS**:2 points) Now, try to think of another way of estimating the value of Pi using the same Monte Carlo approach. You should first describe your idea (can use figures), and then write out the python code for it.

*(1): count dots in the sphere of a cube*

```
samples = 1000000
count = 0
for i in range(samples):
    p = np.random.uniform(size=(3,))
    if (p[0]**2 + p[1]**2 + p[2]**2) < 1:
        count += 1
PI = count/samples*6
```

*(2): discretize the unit square into fine-grids to approximate!*

*(3): Buffon's needle problem*



-------------------------------------------------------------------end-------------------------------------------------------------------