National University of Singapore
School of Computing
CS1010S: Programming Methodology

Mock Midterm

You have only **1 hour and 45 minutes** to solve all **FOUR (4) questions**.
The maximum attainable score is **75**. Good luck!

Prepared by Russell Saerang.

Question 1: Python Expressions [25 marks]

There are several parts to this problem. Answer each part **independently and separately**. In each part, the Python snippet is entered into a Python script and then run. Determine the response printed by the interpreter (Python shell) and **write the exact output**. If the interpreter produces an error message, or enters an infinite loop, explain why and **clearly state the responsible evaluation step**.

A.
```
x, y = 3, 12
def f(x, y):
    x += y
    return x * y
print(f(y, -x))
```

[5 marks]

B.
```
p, q = (), (7, 4, 2, 5, 3)
for r in q:
    if r % 5 == 2:
        p = (r,) + p
    else:
        p += (r - 2,)
print(p)
```

[5 marks]

C.
```
m, p = "mutton", "python"
mp = p[:4] + m[-2:]
if p not in mp:
    print("yum")
else:
    print("hum")
if "on" in p:
    print("bum")
elif mp:
    print("tum")
```

[5 marks]

D.
```
n = 420
while n >= 1:
    n //= 3
    if n % 2 == 0:
        print(2*n)
        continue
    elif n % 7 == 1:
        print(n//2)
        break
    n -= 10
```

[5 marks]

E.
```
def con(x):
    return lambda y: y(y(x))
def fuse(x):
    return lambda y: x(x(y))
much = lambda x: x+1
d = lambda x: 4*x
print(con(fuse(d)(2))(much))
```

[5 marks]

Question 2: Binomial Distribution [19 marks]

Binomial distribution is a commonly used probability distribution, especially in the field of statistics and probability. It is a type of discrete probability distribution of a random variable X where X is the number of successes in a series of trials with two given parameters, n and p, the number of trials and the probability of succeeding a single trial, respectively.

The probability of having k successes in n trials can be denoted as

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Suppose we are to implement a function `choose` that takes in two nonnegative integer inputs n, k and returns

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

i.e. the number of ways to pick k items out of n items.
For the following questions you may assume that $k \leq n$.

A. Provide a recursive implementation of the function `choose`. For example, `choose(10, 3)` will return `120`. **Note that you are not allowed to use any additional functions**.
Hint: Suppose we have m items left to pick. Divide into two cases whether you picked the k-th item or not. How many items left to choose and how many items remaining that can be chosen for each cases?

[4 marks]

B. State the order of growth in terms of time and space for the function you wrote in Part (A). Briefly explain your answer.

[3 marks]

C. Provide an iterative implementation of the function `choose`. **Note that you are not allowed to use any additional functions**.
Hint: Note that you can express $\binom{n}{k}$ like the following.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{1 \cdot 2 \cdot \ldots \cdot k} = \frac{n}{1} \cdot \frac{(n-1)}{2} \cdot \ldots \cdot \frac{(n-k+1)}{k}$$

[4 marks]

D. State the order of growth in terms of time and space for the function you wrote in Part (C). Briefly explain your answer.

[2 marks]

E. Now that we have defined the function `choose`, we can find the probability that the random variable X has a value less than or equal to some nonnegative integer k. Mathematically speaking,

$$\begin{aligned} P(X \leq k) &= P(X = 0) + P(X = 1) + \ldots + P(X = k) \\ &= \binom{n}{0} p^0 (1-p)^{n-0} + \binom{n}{1} p^1 (1-p)^{n-1} + \ldots + \binom{n}{k} p^k (1-p)^{n-k} \end{aligned}$$

The function `cumulative_prob` takes in a nonnegative integer for the number of trials, a real number as probability, and an integer representing the number of successes. It returns the cumulative probability of the random variable X being less than or equal to the number of successes.

For example, **`cumulative_prob(10, 0.5, 3)`** returns **`0.171875`** because it computes the value of $P(X \leq 3)$ where X is a binomial random variable obtained from 10 trials and success probability of 0.5. This means $n = 10$, $p = 0.5$, and $k = 3$.

$$\begin{aligned}
P(X \leq 3) &= P(X = 0) + P(X = 1) + P(X = 2) + P(X = 3) \\
&= \binom{10}{0} 0.5^0 (1 - 0.5)^{10-0} + \binom{10}{1} 0.5^1 (1 - 0.5)^{10-1} \\
&\quad + \binom{10}{2} 0.5^2 (1 - 0.5)^{10-2} + \binom{10}{3} 0.5^3 (1 - 0.5)^{10-3} \\
&= \left(\binom{10}{0} + \binom{10}{1} + \binom{10}{2} + \binom{10}{3}\right) (0.5)^{10}s \\
&= (1 + 10 + 45 + 120)(0.5)^{10} \\
&= \frac{176}{1024} = \boxed{0.171875}
\end{aligned}$$

Using the previously defined functions, provide an implementation to the function `cumulative_prob`. You may assume that the given input will be a valid input, e.g. the probability given will lie in the $[0, 1]$ interval.
Hint: Remember that

$$P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

for any nonnegative integer $k \leq n$. Since we have implemented $\binom{n}{k}$ with `choose`, we can calculate $P(X \leq k)$ right away!

[4 marks]

F. Is your implementation in Part (E) recursive or iterative? State the order of growth in terms of time and space of your implementation. Briefly explain your answer.

[2 marks]

Question 3: Higher Order Distribution [10 marks]

A. **[HARD]** Consider the higher-order function `fold` which was taught in class.

```
def fold(op, f, n):
    if n == 0:
        return f(0)
    else:
        return op(f(n), fold(op, f, n-1))
```

The function `choose` defined in Question 2 can be defined in terms of `fold` as follows:

```
def choose(n, k):
    <PRE>
    return int(fold(<T1>, <T2>, <T3>))
```

Please provide possible implementations for the terms **T1**, **T2**, and **T3**. You may optionally define other functions in **PRE** if needed.

Note: You are to use the higher-order function and not solve it recursively or iteratively or use a formula.

[6 marks]

B. Consider the higher-order function `sum` which was taught in class.

```
def sum(term, a, next, b):
    if a > b:
        return 0
    else:
        return term(a) + sum(term, next(a), next, b)
```

It turns out that the function `cumulative_prob` defined in Question 2 can also be defined in terms of `sum` as follows:

```
def cumulative_prob(n, p, k):
    <PRE2>
    return sum(<T4>, <T5>, <T6>, <T7>)
```

Please provide possible implementations for the terms **T4**, **T5**, **T6**, and **T7**. You may optionally define other functions in **PRE2** if needed.

Note: You are to use the higher-order function and not solve it recursively or iteratively or use a formula.

[4 marks]

Question 4: Brawl Stars! [21 marks]

INSTRUCTIONS: Please read the entire question clearly before you attempt this problem!! You are also not to use any Python data types which have not yet been taught in class.

Background [OK to skip] Brawl Stars is a multiplayer online battle arena and third-person hero shooter video game developed and published by the Finnish video game company Supercell. It was released worldwide on December 12, 2018 on iOS and Android. The game features various game modes, each with a different objective. Players can choose from a selection of Brawlers, which are characters that can be controlled with on-screen joysticks in a game match. In Brawl Stars, players battle against other players or AI opponents in multiple game modes. Players can choose between characters called Brawlers that they have unlocked through Boxes, the Brawl Pass, the Trophy Road, or purchased through the Shop to use in battles.

Source: Wikipedia

You are so addicted to Brawl Stars you decided to implement a Brawl Stars account in Python. You hope that implementing such thing can help you study CS1010S and play Brawl Stars at the same time.

Your first step is to design an account data type that stores the amount of coins the account currently has and the brawlers (Brawl Stars characters) unlocked along with the power points obtained for each brawler. Your account data type should support the following functions:

- `make_account` takes in no parameters and returns a new fresh account with 1000 coins at the start.
- `add_coins(account, amt)` returns a new account with **`amt`** coins added to the account.
- `obtain_powerpoint(account, brawler, points)` returns a new account with **`points`** power points added to **`brawler`**'s statistics. If **`brawler`** hasn't been unlocked before, add it to the collection. However, **2 coins** are deducted for each power point.
- `get_coins(account)` returns the amount of coins the **`account`** currently has.
- `get_brawlers(account)` returns a tuple of all the existing brawlers' names from the **`account`**.
- `get_level(account, brawler)` returns the level of the **`brawler`** in the **`account`** based on the following table. If the **`brawler`** does not exist in the **`account`**, assume it has 0 power points.

Level	Points
0	0
1	1-19
2	20-49
3	50-99
4	100-199
5	200-549
6	550

Example execution:

```
>>> acc = make_account()
>>> acc2 = add_coins(acc, 300)
>>> acc3 = obtain_powerpoint(acc2, "Shelly", 125)
>>> get_coins(acc3)
1050
>>> get_level(acc3, "Shelly")
4

>>> acc4 = obtain_powerpoint(acc3, "Colt", 300)
>>> acc5 = obtain_powerpoint(acc4, "Shelly", 75)
>>> get_brawlers(acc5)
('Shelly', 'Colt')

>>> get_level(acc5, "Shelly")
5
>>> get_level(acc5, "Poco")
0
```

A. Decide on an implementation for the account object and implement `make_account`. Describe how the state is stored in your implementation for **`acc5`**.

[3 marks]

B. Implement the function `add_coins(account, amt)`.

[2 marks]

C. Implement the function `obtain_powerpoint(account, brawler, points)`. You may assume that the power points that you have to pay for will not exceed your current coins balance.

[4 marks]

D. Implement the function `get_coins(account)`.

[2 marks]

E. Implement the function `get_brawlers(account)`. The alphabetic order of the brawlers is not important.

[3 marks]

F. Implement the function `get_level(account, brawler)`.

[3 marks]

G. **[Cash Overflow!]** It turns out that at some point, you might have more than 550 power points in a single brawler. You want to prevent a brawler from having more than 550 points, so the excess power points will simply be converted back into **2 coins** for each power point. For example, if you initially have a brawler with 400 power points and you obtain another 200 power points, you will max out that brawler and get 100 extra coins as a "refund".

Modify the `obtain_powerpoint` function such that it can cater to this problem. Again, may assume that the power points that you have to pay for will not exceed your current coins balance.

[4 marks]

Appendix

The following are some functions that were introduced in class. For your reference, they are reproduced here.

```python
def sum(term, a, next, b):
    if a > b:
        return 0
    else:
        return term(a) + sum(term, next(a), next, b)

def product(term, a, next, b):
    if a > b:
        return 1
    else:
        return term(a) * product(term, next(a), next, b)

def fold(op, f, n):
    if n == 0:
        return f(0)
    else:
        return op(f(n), fold(op, f, n-1))

def enumerate_interval(low, high):
    return tuple(range(low,high+1))

def map(fn, seq):
    if seq == ():
        return ()
    else:
        return (fn(seq[0]),) + map(fn, seq[1:])

def filter(pred, seq):
    if seq == ():
        return ()
    elif pred(seq[0]):
        return (seq[0],) + filter(pred, seq[1:])
    else:
        return filter(pred, seq[1:])

def accumulate(fn, initial, seq):
    if seq == ():
        return initial
    else:
        return fn(seq[0], accumulate(fn, initial, seq[1:]))
```