# PE1 - past paper

Programming Methodology (National University of Singapore)

# CS1010E: Programming Methodology

## PE1 (2020/2021 Sem 1)

### Files

- `PE1.pdf`
- `Question1.py`
- `Question2.py`
- `Question3.py`

### Coursemology

- `Past PE1 > CS1010E 2020/21 Sem 1`

### Questions



## Question 1: Encoding Messages

Harry and Sally are in a relationship and they are both good programmers! Harry likes to send love messages to Sally by encoding them to prevent others from eavesdropping.

The encoding method is very simple. The original message only consists of the 26 uppercase English alphabets and spaces. Harry encodes each alphabet into a two digit number (as `str`) by `"A"`→`"00"`, `"B"`→`"01"`, ⋯, `"Z"`→`"25"`. Finally, a space is represented by 99 (*i.e.*, `" "`→`"99"`).

Below is the brief reminder of the terminologies in Latin alphabets:

- Uppercase: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
- Lowercase: `abcdefghijklmnopqrstuvwxyz`
- Vowels: `aiueo`
- Consonants: `bcdfghjklmnpqrstvwxyz`
- Numeric: `0123456789`

### General Restrictions

You are NOT allowed to use the the following Python ***built-in*** string (`str`) or list (`list`) methods/functions in Question 1:

- `encode()`
- `decode()`
- `replace()`
- `partition()`
- `rpartition()`
- `chr()`
- `split()`
- `rsplit()`
- `translate()`
- `map()`
- `count()`
- `ord()`
- `find()`
- `join()`
- `rfind()`
- `rsplit()`
- `rstrip()`
- `index()`
- `strip()`
- `sort()`
- `sorted()`
- `pop()`

*continue on the next page...*

## 1.1 Iterative Encoding [15 marks]

### Question

Write the *iterative* function `encode_I(word)` to encode the given word (`str`) from English to numeric string (`str`). You may assume that the word will only consists of uppercase characters.

### Restrictions

- You may not use recursive function(s) to solve this.

### Assumptions

- `word` is in uppercase.

### Note

- The output is a string (`str`) instead of an integer (`int`). As such, the leading zeroes are printed.

### Sample Run #1

```
>>> encode_I('HI SALLY')
0708991800111124
```

### Sample Run #2

```
>>> encode_I('I MISS YOU')
08991208181899241420
```

### Sample Run #3

```
>>> encode_I('ARE YOU FREE FOR DINNER TONIGHT')
00170499241420990517040499051417990308131304179919141308060719
```

## 1.2 Recursive Encoding [15 marks]

### Question

Write the *recursive* function `encoding_R(word)` to encode the given word (`str`) from English to numeric string (`str`). You may assume that the word will only consists of uppercase characters.

### Restrictions

- You may not use iterative constructs (*e.g.*, loop, list comprehensions, *etc.*) to solve this.
- The function `encoding_R` must be *recursive* (*i.e.*, it calls itself). The use of any recursive helper functions will not be counted as being recursive.

*continue on the next page...*

### Assumptions

- `word` is in uppercase.

### Note

- The output is a string (`str`) instead of an integer (`int`). As such, the leading zeroes are printed.

### Sample Run #1

```
>>> encoding_R('HI SALLY')
0708991800111124
```

### Sample Run #2

```
>>> encoding_R('I MISS YOU')
08991208181899241420
```

### Sample Run #3

```
>>> encoding_R('ARE YOU FREE FOR DINNER TONIGHT')
00170499241420990517040499051417990308131304179919141308060719
```

# Question 2: Decoding Messages

In order to make the message more secure, Harry decided to encrypt his message in a more serious manner. He will first decided an integer as the offset, `offset`. And every alphabet will move forward by d characters in a cyclic manner. For example, if the offset `offset = 3`, then `"A"`→`"D"`, `"B"`→`"E"`, `"C"`→`"F"`, ⋯, `"W"`→`"Z"`, `"X"`→`"A"`, `"Y"`→`"B"`, `"Z"`→`"C"`. Spaces will remain as spaces (*in other words*, `" "`→`" "`). Then he will send his message by the offset message to Sally. For example, if the message is `"HI SALLY"` and the offset is 5 (*i.e.*,`offset = 5`). Then

`"HI SALLY"`→`"MN XFQQD"`→`"1213992305161603"`

However, you do NOT have to code this offset encoding above.

## General Restrictions

You are NOT allowed to use the the following Python ***built-in*** string (`str`) or list (`list`) methods/functions in Question 2:

- `encode()`
- `decode()`
- `replace()`
- `partition()`
- `rpartition()`
- `chr()`
- `split()`
- `rsplit()`
- `translate()`
- `map()`
- `count()`
- `ord()`
- `find()`
- `join()`
- `rfind()`
- `rsplit()`
- `rstrip()`
- `strip()`
- `sort()`
- `sorted()`
- `pop()`

## 2.1 Offset Decoding [25 marks]

Sally knows that the message she received is encoded with an offset `offset`.

### Question

Write a function `decode(msg, offset)` that will decode the message `msg` (`str`) knowing that the message is offset by `offset` (`int`).

### Assumptions

- `msg` will be numeric string (`str`).

### Sample Run #1

```
>>> decode('1213992305161603',5)
HI SALLY
```

### Sample Run #2

```
>>> decode('000715991910239917071399120722191 7',123)
HOW ARE YOU TODAY
```

### Sample Run #3

```
>>> decode('1900089912130006059924161605990025991100002499050025201819 05',12)
HOW ABOUT MEET ON ZOOM TONIGHT
```

## 2.2 Secret Decoding [10 marks]

However, Harry will NOT send the offset to Sally. Sally has to guess what that is. One of the clue is that Sally knows the message from Harry will always include `"LOVE"` as a substring in the message.

### Question

Write a function `decode_with_love(msg)` that will decode the message `msg` (`str`) without knowing the offset and return the decoded message with the word `"LOVE"` in it.

### Assumptions

- `msg` will be numeric string (`str`).
- The original message will always have the word `"LOVE"` in it.

### Sample Run #1

```
>>> decode_with_love('090619069902190699210906992016150820992406991316230 6')
HERE ARE THE SONGS WE LOVE
```

### Sample Run #2

```
>>> decode_with_love('011607111713129925120299011013200316')
CRIMSON AND CLOVER
```

### Sample Run #3

```
>>> decode_with_love('0819199906220299211212119916009919220312')
ALL YOU NEED IS LOVE
```

# Question 3: Famous Paintings

## 3.1 Simple Area [25 marks]

We know this famous painter Piet Mondrian whose paintings are partitioned into rectangles of colours like red, blue and yellow. We would like to draw something similar with **white**, **yellow** and **red**. Here is an example of our paintings:

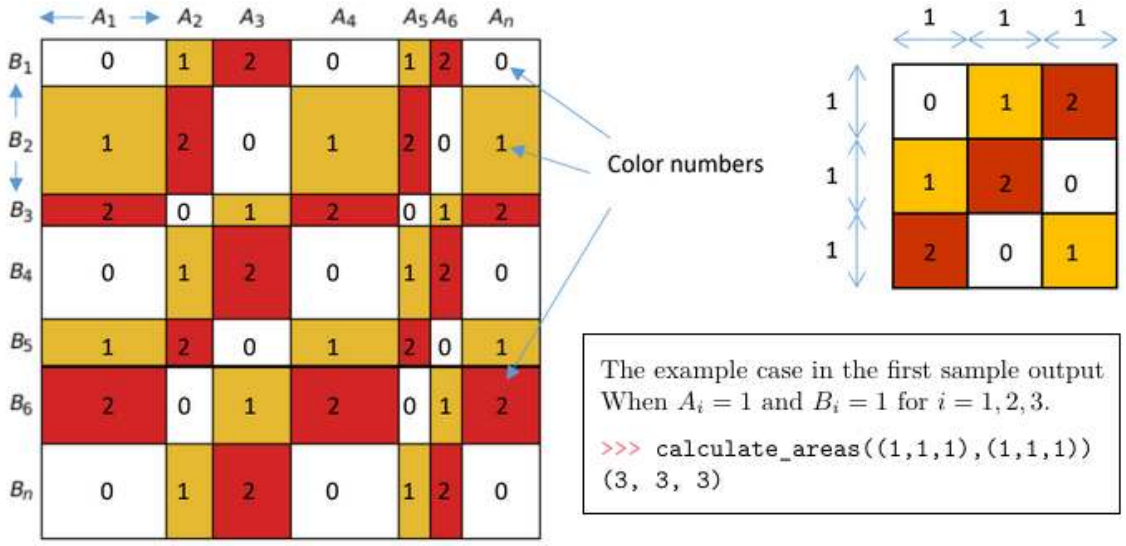


Figure 1: white = color number 0, *yellow* = color number 1 and *red* = color number 2.

We partitions the painting into vertical strips of width $A_1$, $A_2$, $A_3$, ..., $A_n$ and horizontal strips of height $B_1$, $B_2$, $B_3$, ..., $B_n$ centimetres for some positive integer $n > 0$. These stripes split the painting into $n \times n$ rectangles. The intersection of vertical stripe $i$ and horizontal stripe $j$ has colour number $(i + j - 2) \mod 3$ for all $1 \leq i, j \leq n$. To prepare the painting, we want to know how much paint we need for each colour. We will measure it by the area of each colour in square centimetres.

The widths and heights of the vertical and horizontal strips in centimetres will be given in two sequences:

1. `w_list= [A1, A2, A3, ..., An]`
2. `h_list= [B1, B2, B3, ..., Bn]`

You can assume `len(w_list)` is equal to `len(h_list)` and the numbers in both of the lists are integers (`int`) that are greater than zero. Note that we place `Ai` into the list with index `i - 1`, *e.g.*, `A1` is placed at the position with index `0`. You can assume both lists are not empty and have the same length, and contain positive integers.

### Question

Write the function `calculate_areas(w_list, h_list)` that takes in an input sequence `w_list` and `h_list` of integer (`int`) containing `[A1, A2, A3, ..., An]` and `[B1, B2, B3, ..., Bn]` respectively. The function should return a tuple (`tuple`) that contains the areas of the colour white, yellow and red (*in that order*).

### Restrictions

- You are not allowed to modify the input sequence.

**Assumptions**

- `w_list` and `h_list` can be any sequence (*i.e.*, list (`list`) or tuple (`tuple`)) that contains only positive integer (`int`) greater than 0 (*i.e.*, `seq[i] > 0`)
- `len(w_list) > 0` and `len(h_list) > 0`

**Notes**

- Since the input can be any sequence, your code should work for either kind.

**Sample Run #1**

```
>>> calculate_areas((1,1,1),(1,1,1))
(3, 3, 3)
```

**Sample Run #2**

```
>>> l1 = [6,2,4,5,1,1,4]
>>> l2 = [2,5,1,4,2,3,4]
>>> calculate_areas(l1,l2)
(197, 155, 131)
```

**Sample Run #3**

```
>>> l3 = [1]*10
>>> calculate_areas(l3,l3)
(34, 33, 33)
```

## 3.2 Efficient Area [10 marks]

You will immediately get this mark if your answer for Question 3.1 can finish within 1 second for the very large inputs such as in sample runs below:

**Sample Run #1**

```
>>> l4 = [i for i in range(100000)]
>>> calculate_areas(l4,l4[::-1])
(8333166669722177778, 8333166666388911111, 8333166666388911111)
```

**Sample Run #2**

```
>>> l4 = [i for i in range(100000)]
>>> calculate_areas(l4,l4)
(8333166668611088889, 8333166665277822222, 8333166668611088889)
```

**– End of Paper –**