# NATIONAL UNIVERSITY OF SINGAPORE

**CS1010S—Programming Methodology**

2022/2023 Semester 2

Time Allowed: 2 hours

---

## INSTRUCTIONS TO STUDENTS

1. Please write your Student Number only. Do not write your name.
2. The assessment paper contains **FIVE (5) questions** and comprises **FOURTEEN (14) pages** including this cover page.
3. Weightage of each question is given in square brackets. The maximum attainable score is 100.
4. This is a **CLOSED** book assessment, but you are allowed to bring **ONE** double-sided A4 sheet of notes for this assessment.
5. All questions must be answered within the box provided in the **ANSWER SHEET**. **Anything written outside the answer box will not be accepted.**
6. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red or green colour).
7. **Marks may be deducted** for i) unrecognisable handwriting, and/or ii) excessively long code. Where dotted lines are drawn in the answer box, at most one line of code can be written on each dotted line. **Your answer must fit within the number of lines given**. All corrections should be cleanly erased or covered with correction fluid or tape.
8. You may apply function abstraction by reusing functions defined in earlier parts of the question.
9. Common `List` and `Dictionary` methods are listed in the Appendix for your reference.

This page is intentionally left blank.

It may be used as scratch paper.

# Question 1: Python Expressions [30 marks]

There are several parts to this problem. Answer each part **independently and separately**.

In each part, one or more Python expressions are entered into the interpreter (Python shell). Determine the printed output and **write the exact output inside the answer box**. If the interpreter produces an error message, or enters an infinite loop, explain why and clearly **state the responsible evaluation step**.

**A.** [5 marks]

```
i = 0
while i < 5:
    j = i
    while j < 5:
        if i < j:
            print(j - i)
        j += 2
    i += 1
print(j)
```

**B.** [5 marks]

```
d1 = {1: [1,1], 2: (2, 4), 3:'ab', 4: [8, 4], 5: 'ac'}
d2 = {}
for k, v in d1.items():
    if k % 2:
        d2[v[0]] = d1[k][1]
    else:
        d2[v[1]] = d1[k][0]
print(d2)
```

**C.** [5 marks]

```
class A:
    def fun(self, num):
        print("Fun in A")
        if num == 2:
            return num
        else:
            return num + self.fun(num-1)
class B(A):
    def fun(self, num):
        print("Fun in B")
        return super().fun(num)
print(B().fun(3))
```

**D.** [5 marks]

```
a = [1, 2, [1, 2]]
b = [1, 2, a[:2]]
print(a[-1] in b)

c = a.copy()
b[-1][-1] = 3
print(c)

c[-1][-1] = 3
print(a)

a.append(b[-1])
print(a)

a[-1][-1] = 1
print(b)
```

**E.** [5 marks]

```
s = "abcde"
try:
    t = s[:-2]
    print(t)
    t[0] = s[1]
    print(t)
except TypeError:
    t = t + s[1]
    print(t)
except IndexError:
    t = t + s[0]
    print(t)
finally:
    print(t + s[-1])
    print(t[6])
```

**F.** [5 marks]

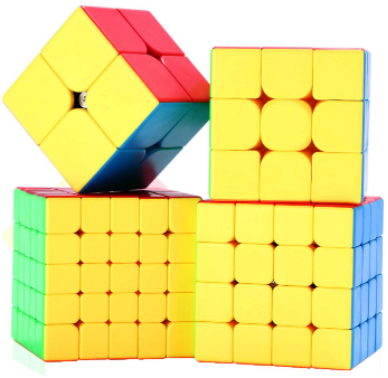
```
def foo(a):
    return lambda n: [a] * n

def add(x, y):
    return x + y

print(foo(3)(2))
print(foo(foo(3)(2))(2))
print(add(*(3, 2)))
print(add(*foo(3)(2)))
print(add(*foo(foo(3)(2))(2)))
```

# Question 2: Rubik's Cube [24 marks]

A Rubik's Cube is a puzzle consisting of a cube with coloured faces made of 26 smaller coloured blocks attached to a spindle in the centre, allowing each face to of the cube to rotate freely. While the original classic cube invented by Ernő Rubik was a $3 \times 3 \times 3$ cube, different sizes of the cube puzzle are also possible.



**Rubik's cubes of different sizes.**

Each face of a Rubik's Cube can be represented using a square matrix, which in Python, can be represented using the typical list-of-lists. Each row of the matrix is a list, and an outer list contains the rows, from the top to the bottom.

Note that while the face of the classic Rubik's Cube is $3 \times 3$, your implementation should work with cubes of any arbitrary size.

**A.** Ben wrote the following function `make_sq_matrix` to help construct a $n \times n$ square matrix and fill it with a given value:

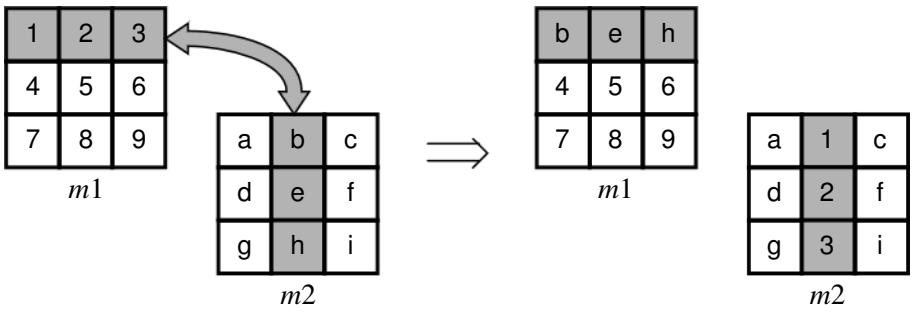
```
>>> def make_sq_matrix(n, value):
...     return [[value] * n] * n

>>> face = make_sq_matrix(3, 1)
>>> face
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
```

On first look it seems that the function works. However, a quick test reveals there is a problem with the returned matrix.

Draw a box-pointer diagram for the matrix `face` and give a correct implementation of the function `make_sq_matrix`. [4 marks]

**B.** It is helpful to have a function to swap a row of a matrix with a column of another matrix. For example:

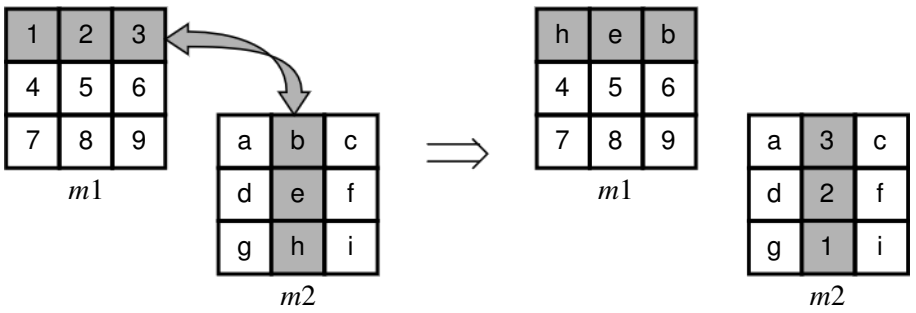



**Two matrices $m1$ and $m2$ swap their row 0 and column 1, respectively.**

Implement the function `swap_rowcol` which takes as input a matrix $m1$ and a row, and another matrix $m2$ and a column. The function swaps the given row of $m1$ with the given column of $m2$.

The row and column index follows the Python indexing scheme, namely non-negative indices counts from the front and negative counts from the back. You may assume both matrices are square matrices of the same size. [4 marks]

**C.** Sometimes, when exchanging the row and columns of two matrices, we require the row and column be flipped. In other words, the row and column are reversed after a swap. For example:

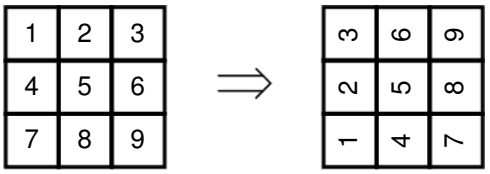



**Two matrices $m1$ and $m2$ have their respective row 0 and column 1 flipped.**

Implement the function `flip_rowcol` which takes as input a matrix $m1$ and a row, and another matrix $m2$ and a column. The function swaps and reverses the row and column of matrix $m1$ and $m2$, respectively. Your implementation **must** make use of the `swap_rowcol` function defined earlier.

Hint: Reversing a row in a matrix can be easily done with a `list` operation. [4 marks]

**D.** Rotating a face counter-clockwise can now be easily done by flipping the rows of the matrix with the columns.




**Matrix rotated counter-clockwise**

However, this operation cannot be done in-place, i.e. on the same matrix. A temporary square matrix has to be used to perform the flip.

Implement the function `rotate_ccw` that takes in a square matrix and modifies it by rotating it counter-clockwise. Your implementation **must** use the function `flip_rowcol` defined earlier.

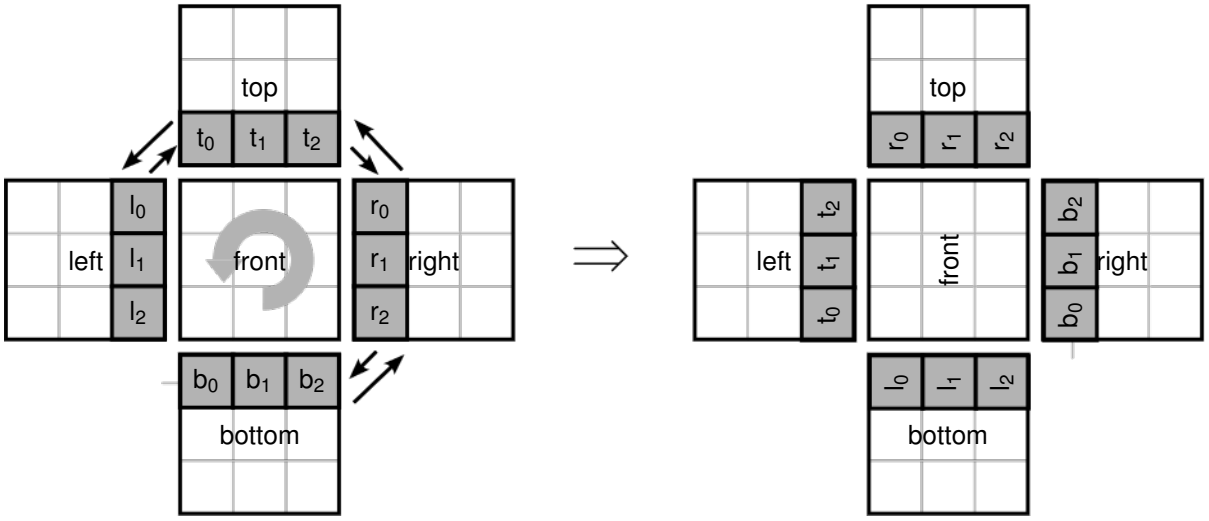
[6 marks]

**E.** A Rubik's Cube can be represented as a collection of faces. For example,

```
cube = [top, left, front, right, back, bottom]
```

All faces are square matrices of the same size and contains values representing the colour of the cube. The faces are oriented according to how they would appear when viewed head-on to a person.

When a face is rotated, the adjoining edge of the 4 adjoining faces are also exchanged. For example, when rotating the front face 90 degrees counter-clockwise, the rows and columns of the top, left, bottom and right faces are exchanged, as shown:




**Front face of a Rubik's Cube rotated 90 degrees counter-clockwise.**

In fact, instead of 4 simultaneous exchanges, the rotation is a result of 3 consecutive exchanges: left ⇔ top, followed by top ⇔ right, and finally right ⇔ bottom. In other words, the rotation can be performed by only using the operations defined in the previous parts.

Implement the function `rotate_front_ccw` which takes in the six faces of a Rubik's Cube, and updates the faces to the state of the cube after having its front face rotated 90 degrees counter-clockwise. Your implementation **must** only contain function calls of functions defined in the earlier parts. [6 marks]

# Question 3: Library Revisited [24 marks]

In the midterm test, we helped a librarian manage records of available and borrowed books. Our implementation was unwieldy since we used immutable tuples. This time, we will use Python dictionaries that are not only mutable but also efficient for the lookup.

As before, you are provided with an immutable data type `Book` along with the selector functions `get_name` and `get_ncopies` to obtain the name and the number of copies of the book owned by the library.

```
>>> get_name(book1)      >>> get_name(book2)      >>> get_name(book3)
"Harry Potter"           "Twilight"              "The Firm"
>>> get_ncopies(book1)   >>> get_ncopies(book2)   >>> get_ncopies(book3)
5                        1                       2
```

Every librarian maintains a `ledger` that contains a record of currently borrowed books. The ledger is updated whenever a book is borrowed or returned. A dictionary is used to model the ledger. The keys are the borrowed `Book`s, and the value is another dictionary, with name of the borrower (**string**) as its key and the date of issue (DOI) as its value.

An example of a ledger will be like:

```
{ book1: {'Nitya': '10-03-2023', 'Ashish': '12-03-2023', ...},
  book2: {'Nitya': '10-03-2023', 'WaiKay': '20-03-2023', ...},
  ...
}
```

You are provided with an abstraction `get_today()` that returns the current date.

```
>>> DoI = get_today()
>>> DoI
'29-04-2023'
```

We now implement the same functions that we had implemented during the midterm exam.

**A.** Implement the function `is_available` that takes a `ledger` and a `book` as inputs. It returns `True` if the book is available for borrowing, otherwise it returns `False`. [4 marks]

**B.** A book may be borrowed if the borrower has not currently borrowed a copy, and the library has copies of the book that are not yet borrowed.

The function `borrow_book` takes a `ledger`, name of the borrower and, a `book` as inputs. If the `book` is able to be borrowed, the ledger is updated with the name of the borrower and the date of issue. Otherwise, it prints the message `'Unable to borrow!'` without making any update.

Implement the function `borrow_book` that updates the ledger to reflect the status of the books in the library. You may assume that borrowers have unique names. [6 marks]

**C.** The function `return_book` takes a `ledger`, name of the borrower and a `book` as inputs. If the `book` is borrowed by the borrower, the respective entry should be removed from the ledger and the function returns `True`. Otherwise, `False` is returned.

In addition, according to the library policy, a book can not be borrowed for more than 14 days. If the book is returned after 14 days, the function additionally prints `'Book is returned late'` to intimate the borrower.

You are provided with an abstraction `days_between` that returns the number of days between two days.

```
>>> DoR = get_today()
>>> DoR
'01-05-2023'
>>> days_between(DoI, DoR)
2
```

Implement the function `return_book` that updates the ledger accordingly when a borrower returns a book. [6 marks]

**D.** The librarian does not like the current representation of the ledger since it does not allow her to efficiently look for borrowers and books borrowed by them. She needs a new *inverted* ledger with the name of the borrower as its key. The value for every borrower is another dictionary with the borrowed `Book` as a key and the date of issue as its value.

An example of an inverted ledger looks like:

```
{'Nitya' : {book1: '10-03-2023', book2: '10-03-2023'},
 'Ashish': {book1: '12-03-2023'},
 'WaiKay': {book2: '20-03-2023'},
 }
```

Implement the function `inverse_ledger` that takes `ledger` as the input and return its inverted representation. [6 marks]

**E.** The current representation of a ledger does not allow a borrower to borrow more than one copy of the same book at the same time. What changes in the representation of a ledger would allow a borrower to borrow multiple copies of the same at the same time? Please provide an example for `K1`, and `V1`.

```
{ book1: {<K1> : <V1>, ...},
 ...
}
```

[2 marks]

# Question 4: Staff Salaries! [18 marks]

**Instructions: Your implementations should use OOP practices. Redundant code or bad OOP practices will be penalised.**

Any staff at NUS is either an academic staff, who conducts modules, or an admin staff who handles the administrative aspects. Consider the following modules:

```
class NUSStaff:
    def __init__(self, name):
        self.name = name
        self.base_salary = 3000

    def get_base(self):
        return self.base_salary


class AcadStaff(NUSStaff):
    def __init__(self, name, modules):
        self.modules = modules

    def topup(self):
        return len(self.modules) * 2000


class AdminStaff(NUSStaff):
    def __init__(self, name):
        self.role = None

    def change_role(self, role):
        self.role = role

    def topup(self):
        return role.allowance if role else 0
```

**A.** We run following code and it raises different errors.

```
>>> cs1010s = Module('CS1010S')
>>> ashish = AcadStaff("Ashish", [cs1010s])
>>> linda = AdminStaff("Linda")
>>> john = NUSStaff("John")

>>> ashish.get_base()
Error: ...
>>> linda.get_base()
Error: ...
>>> john.get_base()
3000
```

```
>>> ashish.get_topup()
2000
>>> linda.get_topup()
Error: ...
>>> john.get_topup() # Expected topup for a generic staff is 0
Error: ...
```

Provide the changes in `NUSStaff`, `AdminStaff` and `AcadStaff` implementations that fix the errors above. Please provide the changes and refrain from writing the entire class definitions again. [6 marks]

**B.** We want to implement a `get_salary` function that computes the net salary as a combination of the base salary and the corresponding topups. After the implementation, the function works as follows:

```
>>> ashish.get_salary()
5000
>>> john.get_salary()
3000
>>> linda.get_salary()
3000

>>> hr = Role('HR')
>>> hr.allowance = 1000
>>> linda.change_role(hr)
>>> linda.get_salary()
4000
```

Which classes should provide the implementation of the method `get_salary`? Provide the **minimal** implementation for the method. **You need not implement the `Role` class and may assume its implementation to be available.** [4 marks]

**C.** As the academic staff progresses in the career, they may be appointed as deans and given additional administrative duties. Thus, we can represent them as both academic and administraive staff by subclassing both `AcadStaff` and `AdminStaff` as follows:

```
class AcadAdminStaff(AcadStaff, AdminStaff):
    pass
```

Suppose we instantiate WaiKay as follows:

```
>>> waikay = AcadAdminStaff("WaiKay", [cs1010s])
```

Besides `waikay.name` and `waikay.modules`, what other properties does `waikay` have and what are their values? [2 marks]

**D.** Consider the following code snippet.

```
>>> waikay.get_salary()
5000

>>> dos = Role('Vice-dean of Students')
>>> dos.allowance = 2000
>>> waikay.change_role(dos)
>>> waikay.get_salary()
5000  # 7000 expected
```

There is an incorrect output as WaiKay should get a salary of $7,000 after taking on the Vice-dean role.

Provide a **minimal** fix to the classes to resolve this incorrect behaviour. [6 marks]

# Question 5: 42 and the Meaning of Life [4 marks]

Either: (a) explain how you think some of what you have learnt in CS1010S will be helpful for you for the rest of your life and/or studies at NUS; or (b) tell us an interesting story about your experience with CS1010S this semester [4 marks]

— END OF PAPER —

# Appendix

Parts of the Python documentation is given here for your reference.

## List Methods

- `list.append(x)` Add an item to the end of the list.
- `list.extend(iterable)` Extend the list by appending all the items from the iterable.
- `list.insert(i, x)` Insert an item at a given position.
- `list.remove(x)` Remove the first item from the list whose value is $x$. It is an error if there is no such item.
- `list.pop([i])` Remove the item at the given position in the list, and return it. If no index is specified, removes and returns the last item in the list.
- `list.clear()` Remove all items from the list
- `list.index(x)` Return zero-based index in the list of the first item whose value is $x$. Raises a `ValueError` if there is no such item.
- `list.count(x)` Return the number of times x appears in the list.
- `list.sort(key=None, reverse=False)` Sort the items of the list in place.
- `list.reverse()` Reverse the elements of the list in place.
- `list.copy()` Return a shallow copy of the list.

## Dictionary Methods

- `dict.clear()` Remove all items from the dictionary.
- `dict.copy()` Return a shallow copy of the dictionary.
- `dict.items()` Return a new view of the dictionary's items ( `(key, value)` pairs).
- `dict.keys()` Return a new view of the dictionary's keys.
- `dict.pop(key[, default])` If *key* is in the dictionary, remove it and return its value, else return $default$. If $default$ is not given and *key* is not in the dictionary, a `KeyError` is raised.
- `dict.update([other])` Update the dictionary with the key/value pairs from *other*, overwriting existing keys. Return `None`.
- `dict.values()` Return a new view of the dictionary's values.

Scratch Paper

— H A P P Y  H O L I D A Y S ! —