CS1010S

Tutorial 9: Advanced List Processing & Exception

Nicholas Russell Saerang (russellsaerang@u.nus.edu)

Announcement

This is the last tutorial.

Next week will be a consultation.

Attendance is optional.

Do note that this Saturday, 13 April is your PE. ATB!

Table of contents

Analysis of Program (NOT TESTED)

The Big-O Notation

How to measure the efficiency of code?

There are many ways to measure efficiency, like:

1. How fast
2. How much memory need
3. Whether the output is optimum
4. ...

But generally, we focus on Time and Space (Memory).

Order of Growth

We want to describe how the time (or space) an algorithm needs increases as the size of the input it processes increases.

Think of it as describing how the "effort" of a program scales when the amount of "work" (data) it has to deal with gets bigger.

The precise mathematical description uses the idea of Big-O Notation. (Not in syllabus, you will learn in CS2040)

Multiple Representation

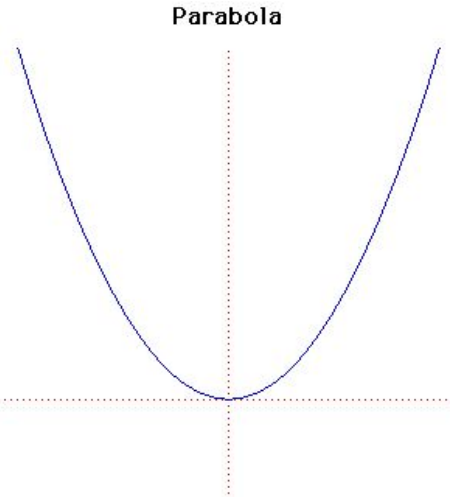
Information representation

Representation of information

How to represent a parabola curve?

x	y
.	.
.	.
.	.
.	.

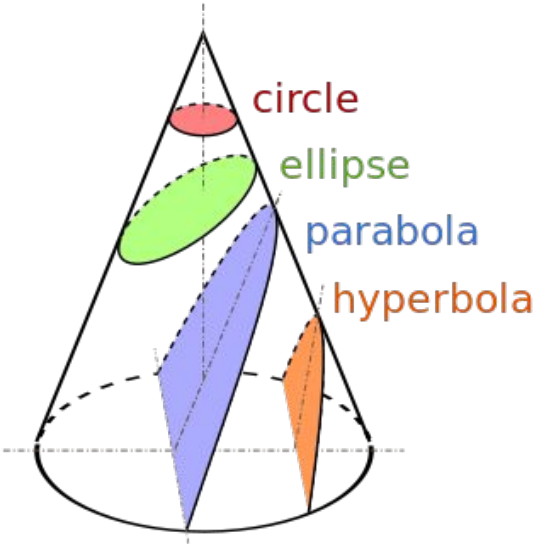
Table



Graph

$$f(x) = ax^2 + bx + c$$

Equation



Conic Session

Representation of information

There are many ways to represent information, and different representation have different benefit.

Similarly, when you are doing programming, you are representing your data using data structure provided by the Language.

And it is important to choose the **suitable (there is no right or wrong)** representation.

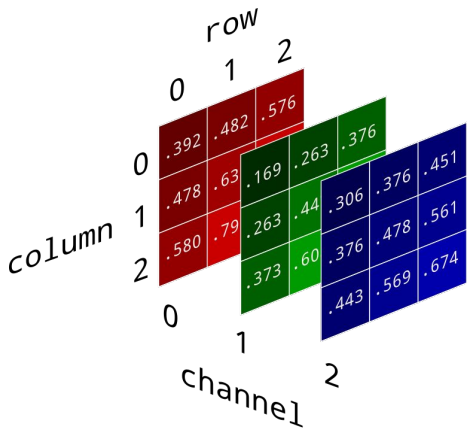
From previous tutorial (the PE question), imagine you represent the data using string/tuple, instead of dictionary... It will be a super tedious process.

Representation of information

Recall your mission 7, when you are working with the image RGB.

There are many ways to represent RGB, here is two example

1) RGB Triplet (which you have been working with)
    a) Use for web color specification (CSS color etc)
    b) Screen Display (your monitor)
2) Multi-channel representation (AY2021/2022 Sem 2 Final Q2)
    a) Easy to do matrix operation`
    b) Quantum Image Representation



Exception

Exceptions

As opposed to error codes, which are obscure…

```
exit(0)    // Success
exit(1)    // Failure
exit(126)  // Permission issue
exit(127)  // Command not found
exit(130)  // SIGINT, Ctrl-C
exit(137)  // SIGKILL
```

… exceptions are much more verbose.

```
PermissionError: [Errno 13] Permission denied:
...
```

Exceptions Hierarchy

```
BaseException
+-- KeyboardInterrupt                        Ctrl-C
+-- Exception
     +-- ArithmeticError
     |    +-- ZeroDivisionError              >>> 2/0
     +-- AttributeError                      >>> "".value
     +-- ImportError                         >>> from math import happiness
     |    +-- ModuleNotFoundError            >>> import cs1010s
     +-- LookupError
     |    +-- IndexError                     >>> [1][2]
     |    +-- KeyError                       >>> {1:2}[3]
     +-- NameError                           >>> del f; f
     |    +-- UnboundLocalError              >>> def f(): print(k); k = 1
     +-- RuntimeError
     |    +-- RecursionError                 >>> def f(): f()
     +--  SyntaxError                        >>> 01
     +--  TypeError                          >>> 1 + "1"
     +--  ValueError                         >>> int("a")
```

Using Exceptions

Raising exceptions for corner cases, i.e. product()

```
def product(nums):
    if len(nums) == 0:
        raise ValueError("nums must have >= 1 element!”)
    if len(nums) == 1: return nums[0]
    return nums[0] * product(nums[1:len(nums):1])
```

Catching exceptions to redirect program flow

```
try:
    r = requests.get("https://google.com/”)
    print(r.status_code)
except KeyboardInterrupt:
    print("GET request terminated.”)
```

Catching Exceptions

```
def typeerr():
    max(unbounderr())
def unbounderr():
    k = [1]
    return k

try:
    typeerr()
    print("Running...")
except TypeError: print("TypeError")
except NameError: print("NameError")
else: print("Safe!")
finally: print("Evaluation completed.")
```

Output:

Catching Exceptions

```
def typeerr():
    max(unbounderr())
def unbounderr():
    k = [1]
    return k

try:
    typeerr()
    print("Running...")
except TypeError: print("TypeError")
except NameError: print("NameError")
else: print("Safe!")
finally: print("Evaluation completed.")
```

Output:

```
Running…
Safe!
Evaluation completed.
```

Catching Exceptions

```
def typeerr():
    max(unbounderr())
def unbounderr():
    k = [1]
    print(k)

try:
    typeerr()
    print("Running...")
except TypeError: print("TypeError")
except NameError: print("NameError")
else: print("Safe!")
finally: print("Evaluation completed.")
```

Output:

Catching Exceptions

```
def typeerr():
    max(unbounderr())
def unbounderr():
    k = [1]
    print(k)

try:
    typeerr()
    print("Running...")
except TypeError: print("TypeError")
except NameError: print("NameError")
else: print("Safe!")
finally: print("Evaluation completed.")
```

Output:

```
[1]
TypeError
Evaluation completed.
```

Catching Exceptions

```
def typeerr():
    max(unbounderr())
def unbounderr():
    print(k)
    k = [1]

try:
    typeerr()
    print("Running...")
except TypeError: print("TypeError")
except NameError: print("NameError")
else: print("Safe!")
finally: print("Evaluation completed.")
```

Output:

Catching Exceptions

```
def typeerr():
    max(unbounderr())
def unbounderr():
    print(k)
    k = [1]

try:
    typeerr()
    print("Running...")
except TypeError: print("TypeError")
except NameError: print("NameError")
else: print("Safe!")
finally: print("Evaluation completed.")
```

Output:

```
NameError
Evaluation completed.
```

Catching Exceptions

```
def typeerr():
    max(unbounderr())
def unbounderr():
    unbounderr()
    k = [1]

try:
    typeerr()
    print("Running...")
except TypeError: print("TypeError")
except NameError: print("NameError")
else: print("Safe!")
finally: print("Evaluation completed.")
```

Output:

Catching Exceptions

```
def typeerr():
    max(unbounderr())
def unbounderr():
    unbounderr()
    k = [1]

try:
    typeerr()
    print("Running...")
except TypeError: print("TypeError")
except NameError: print("NameError")
else: print("Safe!")
finally: print("Evaluation completed.")
```

Output:

```
Evaluation completed.
RecursionError: maximum
    recursion depth exceeded
```

Tutorial 9

Last stretch!

Question 1: Matrix - Transpose

A matrix can be represented in Python by a list of lists (nested lists). For example, `m = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]` represents the following 3×4 matrix:

$$\begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{vmatrix}$$

(a) Write a function `transpose` which takes in a matrix and transposes it. Basically, this converts an $m \times n$ matrix into an $n \times m$ matrix. The function should return a **new** matrix.

(b) Now re-implement `transpose` such that it returns the **original** matrix instead.

Question 1a: Matrix - Transpose

```
def transpose(matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            matrix[i][j], matrix[j][i] \
                = matrix[j][i], matrix[i][j]
    return matrix # any issues?
```

Question 1a: Matrix - Transpose

```
def transpose(matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            matrix[i][j], matrix[j][i] \
                = matrix[j][i], matrix[i][j]
    return matrix # any issues?

# Wrong:
# Assumes square matrix => Index Error
```

Question 1a: Matrix - Transpose

```
def transpose(matrix):
    transposed_matrix = []
    for i in range(0,len(matrix[0]),1):
        new_row = []
        for j in range(0,len(matrix),1):
            new_row.append(matrix[j][i])
        transposed_matrix.append(new_row)
    return transposed_matrix
```

Question 1a: Matrix - Transpose

Some trick in Python.
Use this only if you understand the fundamental, and know how it works.

```
def transpose(matrix):
    return list(map(list, zip(*matrix)))

# Returns some iterator, e.g.
# zip(*[[1,2],[3,4],[5,6]])
# = zip([1,2],[3,4],[5,6])
# = iterator([(1,3,5),(2,4,6)])
```

Question 1b: Matrix - Transpose

```
def transpose(matrix):
    transposed_matrix = []
    for i in range(0,len(matrix[0]),1):
        new_row = []
        for j in range(0,len(matrix),1):
            new_row.append(matrix[j][i])
    # Clear the previous matrix
    matrix.clear()
    # Extend the new matrix to the old one
    matrix.extend(transposed_matrix)
    return matrix
```

Question 1: Matrix - Transpose

(c) Write a function `row_sum` which takes in a matrix and returns a list, where the i-th element is the sum of the elements in the i-th row of the matrix. You can assume that the matrix will not be empty, and has exactly $m \times n$ elements, where m and n are positive integers.

```
>>> row_sum(m)
[10, 26, 42]
```

(d) Write a function `col_sum` which takes in a matrix and returns a list, where the i-th element is the sum of the elements in the i-th column of the matrix.

```
>>> col_sum(m)
[15, 18, 21, 24]
```

Question 1c: Matrix - Row Sum

```
def row_sum(matrix):
    return list(map(sum, matrix))
```

```
def row_sum(matrix):
    res = []
    for row in matrix:
        new_sum = 0
        for ele in row:
            new_sum += ele
        res.append(new_sum)
    return res
```

Question 1d: Matrix - Column Sum

```
def col_sum(matrix):
    result = [0] * len(matrix[0])
    for row in matrix:
        for i in range(len(row)):
            result[i] += row[i]
    return result

def col_sum(matrix):
    return row_sum(transpose(matrix))
```

This one very important

Will come out in your practical exam

Question 2: Gradelist

You are given a list of students in the following form (name, letter grades, score). For example,

```
students = [
    ('tiffany', 'A', 15),
    ('jane', 'B', 10),
    ('ben', 'C', 8),
    ('simon', 'A', 21),
    ('eugene', 'A', 21),
    ('john', 'A', 15),
    ('jimmy', 'F', 1),
    ('charles', 'C', 9),
    ('freddy', 'D', 4),
    ('dave', 'B', 12)]
```

The functions that you write for this question should work with any arbitrary list of students and not just for this sample list.

Question 2a: Gradelist - Mode

(a) Write a function `mode_score` that takes a list of students and returns a list of the mode scores (scores that appear the most number of times). If there is only one score with the highest frequency, then this list would only have one element.

For example:

```
>>> mode_score(students)
[15, 21]
```

Question 2a: Gradelist - Mode

```
def mode_score(students):
    # Store all the scores in some form of seq
    # From the seq, find the most freq score (max_f)
    # Using the max_f, filter the non mode score
    # Then store the unique max_scores in list and return
```

Question 2a: Gradelist - Mode

```
def mode_score(students):
    # Store all the scores in some form of seq
    # From the seq, find the most freq score (max_f)
    # Using the max_f, filter the non mode score
    scores = list(map(lambda s: s[2], students))
    max_f = max(map(lambda s: scores.count(s), scores))
    mode = filter(lambda s: scores.count(s) == max_f, scores)
```

Question 2a: Gradelist - Mode

```
def mode_score(students):
    scores = ...
    max_f = ...
    mode = ...

    # Then store the unique max_scores in list and return
    unique_students = [] # store unique max_scores
    for i in mode:
        if i not in unique_students:
            unique_students.append(i)
    return unique_students
```

Question 2b: Gradelist - Top k

(b) Write a function `top_k` that takes a list of students and an integer k and returns a **new** list of k students with the highest scores in alphabetical order. If there are students in the range $(k+1, \cdots k+i)$ who have the same score as the kth student, include them in the list as well. **Do not modify the original list.**

For example:

```
>>> top_k(students, 5)
[('eugene', 'A', 21), ('simon', 'A', 21), ('john', 'A', 15),
('tiffany', 'A', 15), ('dave', 'B', 12)]

>>> top_k(students, 3)
[('eugene', 'A', 21), ('simon', 'A', 21), ('john', 'A', 15),
('tiffany', 'A', 15)]
```

Question 2b: Gradelist - Top k

```
def top_k(students, k):
    # sort by name, then descending score
    # without modify the original
    copy_students = sorted(students,
                           key=lambda s: (-s[2], s[0]))
```

Question 2b: Gradelist - Top k

```
def top_k(students, k):
    copy_students = ...

    if k == 0 or k > len(copy_students):
        return copy_students[:k]
    cutoff = copy_students[k-1][2] # kth student score
    return list(filter(lambda s: s[2] >= cutoff,
                copy_students))
```

Sorted list	Key value order
(Eugene, A, 21)	(-21, Eugene)
(Simon, A, 21)	(-21, Simon)
(John, A, 15)	(-15, John)
(Tiffany, A, 15)	(-15, Tiffany)

Thank you

All the best!