STMC HKOI Training

Lesson 4: Looping and Array

Tsai Yun Chen

November 9, 2022

Goal today

- Concept of loop
- `while` loop
- `for` loop using `range`
- The for loop `for`
- Basics `list`

Loop: Repeat and repeat ….

- Many times in programming we want the code to run repeatedly until certain conditions are met

- For example:
  - Recieving user input: User might input a wrong value. You would want to keep asking for an input until it's right
  - Reading files: You want to keep reading lines until the end of file
  - Games: You want to keep the main code running until the game ends
  - Searching: Sometimes you use computer to search for answers. You would want the computer to keep searching until the solution / close enough solution is reached

Loop: Repeat and repeat ….

- From the examples above, we see the a looping structure always consist of two parts:
  1. The code inside the code that is looped over
  2. A condition that is checked everytime the loop ran to decide whether the loop should continue
- Example:
  - Recieving user input (code inside loop); Is the answer right (terminate condition)
  - Reading files (code inside loop); Is the end of file reached (terminate condition)
  - Main game code (code inside the loop); Is the game over (terminate condition)
  - Searching for answers (code inside the loop); Is the solution found (terminate condition)

Example: Print first N positive integer

- Let’s write a program that takes in an integer N and print out all positive integers i in range $1 \leq$ i $\leq$ N
- For example:
  - If we enter 1, `{1}` will be printed
  - If we enter 4, `{1,2,3,4}` will be printed
  - and etc.

Example: Print first N positive integer

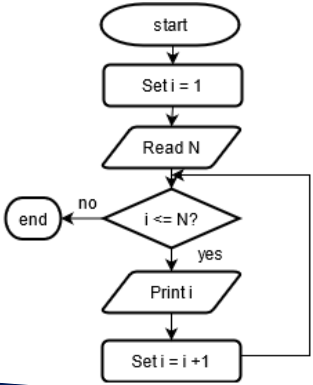
- Some example input and output:

```
$./main          $./main          $./main
5                4                100
1                1                1
2                2                2
3                3                .... /* too long won't list here*/
4                4                99
5                                 100
```

- Problem: How can we implement this in code?

Flow chart

- Let’s look at the flow chart
- Basically, we repeat certain blocks of code until a given condition (in this case i $\leq$ N) is false
- This condition is called the loop condition
- Notice ”Set i = i + 1” is crucial otherwise i will always be smaller than N. This will cause an infinite loop



while loop

- In Python, we can implement that using while loop
- Here is the syntax of while loop

```
while """loop condition""":
  # Remember to indent
  # This will keep looping as long as loop condition is True

# When loop condition is False, the loop will break
# The code will continue to run from here
```

Example: Print first N positive integer

- This is how we write print first N positive integer in python

```python
N = int(input('Enter N: '))
i = 1
while i <= N:
  print(i)   # Print i, remember to indent
  i = i + 1  # This is critical, otherwise infinite loop
print('End of story') # Just some useless print
```

Example: Input validation

- You are writing a registration website for a company.
- In your website, the user is required to enter their age.
- However, some employee of the company might be careless and enter their age incorrectly.
- Write a program that reads in an age, and make sure it's between $18 - 65$ (inclusive)
- If the age is out of this range, prompt the user to renter the information until the input is correct

Example: Input validation

- Example input and ouput:
- Correct input:

```
Enter your age: 18
Ok! Have a nice day!
```

- Incorrect input:

```
Enter your age: 12
Age should be from 18-65
Enter your age: 69
Age should be from 18-65
Enter your age: 27
Ok! Have a nice day!
```

Example: Input validation

- One possible solution:

```python
""" Sample solution for Input validation """

age = int(input('Enter your age: '))
while age < 18 or age > 65:
  print('Age should be from 18-65')
  age = int(input('Enter your age: '))
print('Ok! Have a nice day!')
```

Exercise: Fibonacci number

- Let’s try some exercise
- Fibonacci number is defined such in a way such that first two number are 1, 1
- then starting from the third, it is defined as the sum of previous two
- so it will go like $1, 1, 2, 3, 5, 8, \ldots$
- Now write a program which takes a input n and print the n-th Fibonacci number

for loop

- In principle all loops can be written using while loop
- But sometimes we want to be more concise
- For example, the following loop is clumsy:

```
i = 0
while i < 5:
  print(i)
  i = i+1
```

`for` loop

- In fact, if we want to do loop similar to that above, we can use the `for` loop
- The equivalent for loop for the loop just now is:

```python
for i in range(0,5):
  print(i) # Print numbers 0, 1, 2, 3, 4
```

which looks much nicer

Example: Print first N integer

- Using for loop, our previous example of printing first N integers can be greatly simplified:

```python
""" Print first N integer using for loop """
N = int(input('Enter N: '))
for i in range(0,N):
  print(i+1)
```

General syntax of `for` loop

- In general, the syntax for a `for` loop using `range` is:

```python
for i in range(begin,end,steps):
  # Do things here
```

- This will loop `i` from `begin <= i < end` with `i` increasing by `step` each time it loops
- For example: `range(1,7,1)` will gives you $1, 2, 3, 4, 5, 6$ (notice the last number is excluded)
- Another example: `range(2,9,3)` will give you $2, 5, 8$ (notice each number differ by 3, the step size)

Example: Sum of first n odd numbers

- Write a program using `for` loop that calculate the sum of first n odd numbers

$$S = 1 + 3 + 5 + \cdots + 2n - 1$$

```python
""" Solution: Sum of first n odd numbers """
N = int(input('Enter N: '))
S = 0
for i in range(1,2*N,2): # Upper limit 2N to include 2N-1
  S += i
print('Sum: ',S)
```

Example: Magic triangles

Write a program that recieve an integer n. Print a triangle of height n and base n with using $(*)$. Here are some example outputs

```
>>3           >>5         >>2
*             *           *
**            **          **
***           ***
              ****
              *****
```

(Hint: To print a $*$ without newline, you can use `print('*',end='')`)

Example: Magic triangles+

Modify the program previously to give the following output:

```
>>3            >>5         >>2
*              *           *
**             **          **
***            ***         *
**             ****
*              *****
               ****
               ***
               **
               *
```

Exercise: Magic triangle++

Modify the program previously to give the following output:

```
>>3           >>5          >>2
*             *            *
***           ***          ***
*****         *****        *
***           *******
*             *********
              *******
              *****
              ***
              *
```

List: List of objects

- Loops are useful, but they are most powerful when used with data structures like `list`
- List is also called array in language like C/C++
- A list is an ordered list of objects
- It stores multiple values in a single variable, which we can refer to using an index

List: Example of Lists

- To create a list, we surround some comma-separated values with `[]`
- Let's look at a list to see what exactly it means:

```python
intList = [10,328,321,392] # List of integers

floatList = [40.1,339.2,77.3] # List of floats

strList = ['Billy', 'May', 'Dorian'] # List of strings

boolList = [True,False,True,Flase] # List of booleans

mixedList = [183.3, 282, False, 'Hi'] # List of mixed data types
```

List: Indexing

- Each item in a list is labelled by an index, which we can use to refer to an item
- The indices starts from 0

```
myList = ['Hello',831.9, False, 88]

print('myList[0]: ', myList[0]) # myList[0] = 'Hello'

print('myList[1]: ', myList[1]) # myList[1] = 831.9

print('myList[2]: ', myList[2]) # myList[2] = False

print('myList[3]: ', myList[3]) # myList[3] = 88
```

List: Indexing

- For a list of length `n`, the indices ranges from `0,1,2,...,n-2,n-1`
- Accessing outside this length will results in:
  `IndexError: list index out of range`

```
>> myList = [28,219,3298]

>> myList[3] # Error! Indices from 0 to 2

>> myList[2] # Corret. Get 3298
```

List: Length of list

- The length of list can be obtained by using the `len()` function
- The returned value is an integer
- For example, to get the length of `myList` we write `len(myList)`

```python
myList = ['Hello',831.9, False, 88]

print('Length of list: ', len(myList)) # Length of list: 4
```

List: Add values to end

- We can add values to the end of the list by `append` method
- Syntax: `myList.append(<values>)`

```python
myList = [] # Empty list
print(myList) # Print []

myList.append(3) # Append 3 to list
print(myList) # Print [3]

myList.append('Hi') # Add 'Hi' to the end
print(myList) # Print [3, 'Hi']
```

List: Reading list of inputs

- Let’s say we want to write a program that read in scores of students in a course and see how well they perform
- We can use list to do it

```
studentScore = []
score = 0

while score >= 0: # Keep looping until input -1
  score = float(input('Enter score, enter -1 to terminate:'))
  if score >= 0:
    studentScore.append(score)
```

List: Loop over list

- After reading in data, we can loop the list over with for loop

```python
studentScore = [82,42,72,64,22]

# Print the items in the list
for i in range(0,len(studentScore)):
  print('Student ',i,'score ',studentScore[i])
```

List: Loop over list

- For example, find the largest in the list:

```python
studentScore = [82,42,72,64,22]
largest = studentScore[0]

for i in range(0,len(studentScore)):
  if studentScore[i] > largest:
    largest = studentScore[i] # If we find a score larger than largest, update largest score

print('Highest score: ',largest) # Print highest score
```

List: Loop over list

Exercise: Find minimum

Modify the code above to find the smallest in the list

Exercise: Average score

Write a program that takes scores until −1 is entered, then calculate and output the average score in the group

Exercise: Best student

Write a program that takes in the name and score in two list and output the name of the student with the highest score

Challenge

Sorting

Write a program that takes in a list of N numbers and return a sorted list of the numbers. We will come back to sorting in next slide. You may google for keywords like bubble sort, insert sort or quicksort.

Sorting: naive approach

Let’s take the most naive way to do so, we find the minimum for elements between 1 to n, move it to the head, then do it again and again with fewer elements.

```python
studentScore = [82,42,72,64,22]
for i in range(0,len(studentScore)):
  min=studentScore[i]
  minPos=i
  for j in range(i,len(studentScore)):
    if(studentScore[j]<min):
      min=studentScore[j]
      minPos=j
  studentScore[i],studentScore[minPos]=studentScore[minPos],studentScore[i]
```

But how must time does it takes?

- Let’s have some basic assumption: say each comparison and assignment take constant amount of time, for example each take 0.0001s
- let n be the number of elements we have in the list
- hence we have i goes from 0 to n $- 1$
- for j we start at i but and end at n $- 1$
- for each fixed i, j, we do at most 1 comparison and 2 assignment
- for each i we do also 2 assignment
- to simplify the case, let’s assume we do 5 operations in total for each i, j

Math time

let's just fix some i there, then we consider the time needed for j goes from i to n, but since no matter what j is we do at most 5 operations there, therefore it is

$$\underbrace{5+5+5+5+\ldots+5}_{\text{\# of times for j from i to n}} = 5(n-i+1)$$

Next since i goes from 1 to n, we first observe

- when $i=1$, $5(n-i+1)=5n$
- when $i=2$, $5(n-i+1)=5(n-1)$
- when $i=3$, $5(n-i+1)=5(n-2)$
- when $i=n$, $5(n-i+1)=5$

Therefore, if we sum them all up, we can see that the total time needed will be

$$T=5n+5(n-1)+5(n-2)+\ldots+5=5(1+2+3+\ldots+n)$$

Sequence sum

So now we want to calculate the sum

$$S = 1 + 2 + 3 + ... + n$$

Consider if we pair up the number one by one in the manner that $(1, n)$, $(2, n - 1)$, $(3, n - 2)$ etc. We notice that all these pairs sum up $n + 1$

But how many such pairs we can get? If n is even, then we have $\frac{n}{2}$ such pairs, if n is odd, but then $n - 1$ is even, so we can get $\frac{n-1}{2}$ such pair with an addition n there.

In both case we have the sum

$$S = \frac{n(n+1)}{2}$$

Putting back to our total time we will have

$$T = \frac{5n(n+1)}{2} = \frac{5}{2}n^2 + \frac{5}{2}n$$

Time complexity and some extra notes

When n is large, we can observe that $\mathsf{n}^2 >> \mathsf{n}$, therefore in many cases, we will drop the lower order term and only focus on the leading order, in our cases we will denote

$$\mathsf{T} = \mathsf{O}(\mathsf{n}^2)$$

This is call asymptotic time bound/complexity, it gives a rough bound on how slow the algorithm could be.

Theoretically, general sorting can be improved to be $\mathsf{O}(\mathsf{n} \log \mathsf{n})$, and sometimes $\mathsf{O}(\mathsf{n})$ if we have extra constraint on the data to be sorted.