Python Programming

Iteration Statements

Prof. Chang-Chieh Cheng

Information Technology Service Center

National Chiao Tung University

`while` statements

- **Syntax**

```
while condition:
    indented_statement_block
```

  - Two steps:
    1. If `condition` is True then execute `indented_statement_block` once; Otherwise, stop the `while` statement.
    2. Go back to step 1.

- For example, print the numbers from n to 1.

```
n = int(input('Input a positive integer: '))
while n > 0:
    print(n)
    n = n - 1
```

- Let's try it, print the numbers from 1 to n

`while` statements

- For example, n!, the factorial of n.
  - $n! = 1 \times 2 \times \cdots \times (n-1) \times n$

```
n = int(input('Input a positive integer: '))
fac = 1
while n > 0:
    fac = fac * n
    n = n - 1
print('n! is', fac)
```

`while` statements

- Let's try it
  - given an integer *n* and *n* > 1, design a while loop to compute α and β, where

$$\alpha = 2 \times 4 \times 6 \times \cdots \times (n - n\%2)$$
$$\beta = 1 \times 3 \times 5 \times \cdots \times (n - 1 + n\%2)$$

  - For example, if *n* is 10 then

$$\alpha = 2 \times 4 \times 6 \times 8 \times 10 = 3840$$
$$\beta = 1 \times 3 \times 5 \times 7 \times 9 = 945$$

`while` statements

- Let's try it
  - given an integer *n* and *n* > 1, design a while loop to generate a series of numbers by the following rule.
    1. if *n* is even ➔ *n* = *n* // 2 ➔ print *n*
    2. otherwise ➔ *n* = *n* – 1 ➔ print *n*
    3. repeat 1. until *n* <= 1
  - For example, if *n* is 7, the results will be

    6 3 2 1
  - if n is 16

    8 4 2 1

while statements

- Input loop
  - For example, sum of non-negative numbers

```
n = 0
sum = 0
while n >= 0:
    n = int(input('Input a non-negative integer: '))
    if n >= 0:
        sum = sum + n
print('sum is', sum)
```

  - String appending

```
s = ''
sOut = ''
while s != '.':
    s = input('Input a word: ')
    if s == '.':
        sOut = sOut + s
    else:
        sOut = sOut + ' ' + s
print(sOut)
```

while statements

- Multiple layers of `while` statements
- For example, print a multiplication table

```
x = 1
while x <= 9:
    y = 1
    while y <= 9:
        print(x, ' * ', y, ' = ', x * y)
        y = y + 1
    x = x + 1
```

- Let's try it
  - Modify the above code, such that the results will be

```
1 * 1 = 1    2 * 1 = 2    3 * 1 = 3 ... 9 * 1 = 9
1 * 2 = 2    2 * 2 = 4    3 * 2 = 6 ... 9 * 2 = 18
.
.
.
1 * 9 = 9    2 * 9 = 18   3 * 9 = 27... 9 * 9 = 81
```

`while` statements

- Let's try it
  - Design a two-layer `while` statement to generate the following result:

```
OXOXO
XOXOX
OXOXO
XOXOX
OXOXO
```

  - Furthermore, can you generate any size of the above pattern? For example, 7 x 7, 10 x 10 or 7 x 10.

`while` statements

- Input loop and the nested `while` statement
  - Example:

```
x = 1
while x > 0:
    x = int(input('Input a positive integer: '))
    n = x
    fac = 1
    while n > 0:
        fac = fac * n
        n = n - 1
    print('n! is', fac)
```

  - But 0! will be computed when x is 0

while statements

- break
  - Escape the current while loop a level without any condition.

```
while True:
    n = int(input('Input a positive integer: '))
    if n <= 0:
        break
    fac = 1
    while n > 0:
        fac = fac * n
        n = n - 1
    print('n! is', fac)
```

while statements

- break
  - Notice that using break in a nested while statement

```
while True:
    n = int(input('Input a positive integer: '))
    if n <= 0:
        break
    fac = 1
    while True:
        fac = fac * n
        n = n - 1
        if n == 0:
            break
    print('n! is', fac)
```

`while` statements

- Let's try it
  - Design a while loop that allows a user to input any text until the user input `'quit'` or `'exit'`.

while statements

- continue
  - Skip an iteration rather than stop a while statement

```
while True:
    n = int(input('Input a positive integer: '))
    if n == 0:
        break
    elif n < 0:
        continue
    fac = 1
    while True:
        fac = fac * n
        n = n - 1
        if n == 0:
            break
    print('n! is', fac)
```

`while` statements

- Let's try it
  - Design a while loop that allows a user to input any text until the user input `'quit'` or `'exit'`.
  - But, the user may input many numbers, so you have to sum all input numbers.
    - Use `continue` to skip non-number text.
  - Then, print the sum after the while loop is end.

Exercise

- Modify the example of p.5
- Check whether a positive integer is prime
  - Naive algorithm:
    - An integer is a prime if it is greater than 1 and has no positive divisors other than 1 and itself.
    - For example, 5 is a prime because it can't be divided by 2, 3 , and 4; 6 is not a prime because it can be divided by 2.
  - Quick algorithm
    - If $n = ab$, and $a \leq b$, where n, a, and b are positive integers. Assuming that $a^2 > n$, then $n = ab \geqq a^2 > n$ causes contrary. Therefore, $a^2 \leq n$.
    - So, you can check that if n is prime then there is no any number between 1 and a^2 can divide n, where $a^2 \leq n$.
    - For example, 11 is a prime because it can't be divided by 2 and 3 (you don’t need to test 4 to 10).

`while` statements + list operations

- Using a list to collect input data

```
L = []          # initialization
while True:
    try:
        x = float(input('Input a number: '))
        L.append(x);
    except ValueError:
        break;
print(L)
```

while statements + list operations

- Access each item in a list

```
L = []          # initialization
while True:
    try:
        x = float(input('Input a number: '))
        L.append(x);
    except ValueError:
        break;

i = 0
while i < len(L):
    print(L[i])
    i += 1
```

Exercise

- Design a program that allows user to input several numbers
  - Computing their average
  - Computing their standard deviation
    - The standard deviation of *N* numbers can be computed as follows

$$s = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}(A[i] - \boldsymbol{\alpha})^2}$$

where `A[i]` is the i-th input number, and $\boldsymbol{\alpha}$ is the average of `A[0]` to `A[N-1]`

`for` statements

- Syntax

```
for item in data_sequence:
    indented_statement_block
```

  - Each iteration accesses an item in `data_sequence` (a list or a string), in the order that it appear in the sequence.

  - Example:

```
L = ['dog', 'cat', 'bird']
for pet in L:
    print(pet)
```

for statements

- Let's try it
  - given an integer *n* and *n* > 1, design a while loop to generate a series of numbers by the following rule.
    1. if *n* is even ➔ *n* = *n* // 2 ➔ print *n*
    2. otherwise ➔ *n* = *n* – 1 ➔ print *n*
    3. repeat 1. until *n* <= 1
  - For example, if *n* is 7, the results will be

    6 3 2 1
  - if n is 16

    8 4 2 1
  - But, you have collect these numbers into a list `L`
  - Then, if `L` contains k numbers, let `t = L[k // 2]`
  - Subtract all numbers in `L` by `t`, for example, if `L` contains

    6 3 2 1
  - Then, `t` is 2 and the final results are

    4 1 0 -1

for statements

- range function
  - Generating a list of numbers that over a specified range
  - Three usages
    - range(stop) ➔ 0 to stop - 1
    - range(start, stop) ➔ start to stop - 1
    - range(start, stop, step)
      If step > 0 ➔start, start + step, …, (start + i * step) < stop
      If step < 0 ➔start, start + step, …, (start + i * step) > stop
    - start, stop, and step must be integers
    - step must be non-zero
  - Example:

```
L = list(range(5))
print(L)                # 0, 1, 2, 3, 4
L = list(range(6, 10))
print(L)                # 6, 7, 8, 9
L = list(range(0, 10, 2))
print(L)                # 0, 2, 4, 6, 8
L = list(range(0, -10, -2))
print(L)                # 0, -2, -4, -6, -8
```

`for` statements

- `for` statement with `range` function

```
for x in range(-100, 100, 10):
    print(x)
```

- Range normalization
  - The following code generates a number sequence from -1.0 to 0.99

```
L = []
for x in range(-100, 100):
    L.append(x * 0.01)
print(L)
```

`for` statements

- Using `for` statement to access each item of a list

```
L = [1, 2, 3, 4, 5]
for i in range(len(L)):
    L[i] *= 10
for i in range(len(L)):
    print(L[i])
```

`for` statements

- Let's try it
  - [0, 1] Normalization:
    - Given a list `L` that contains `N` numbers
    - Let `min` is the minimum of `L`, and `max` is the maximum of `L`
    - Then, each number `x` in `L` =
      - `(x - min) / (max - min)`
    - For example, `L = [-10, -2, 0 , 3, 4]`, the results is `[0.0, 0.571, 0.714, 0.929, 1.0]`

sorting

- `sorted` function
  - Ascending sorting

```
L1 = [5, 4, 2, 3, 1]
L2 = sorted(L1)
print(L1)      # [5, 4, 2, 3, 1]
print(L2)      # [1, 2, 3, 4, 5]
```

```
Ls1 = ['cat', 'mouse', 'pig', 'dog', 'bird']
Ls2 = sorted(Ls1)
print(Ls1)     # ['cat', 'mouse', 'pig', 'dog', 'bird']
print(Ls2)     # ['bird', 'cat', 'dog', 'mouse', 'pig']
```

  - The order of characters
    - Based of **ASCII** (American Standard Code for Information Interchange)
    - symbols(`!#$..`) <
      digits(`0-9`) <
      upper-case alphabets(`A-Z`) <
      lower-case alphabets(`a-z`) <
      `{, |, }, ~`

sorting

- `sorted` function
  - Descending sorting

```
L1 = [5, 4, 2, 3, 1]
L2 = sorted(L1, reverse = True)
print(L1)      # [5, 4, 2, 3, 1]
print(L2)      # [5, 4, 3, 2 ,1]
```

```
Ls1 = ['cat', 'mouse', 'pig', 'dog', 'bird']
Ls2 = sorted(Ls1 , reverse = True)
print(Ls1)     # ['cat', 'mouse', 'pig', 'dog', 'bird']
print(Ls2)     # ['pig', 'mouse', 'dog', 'cat', 'bird']
```

sorting

- `sort` method
  - Ascending sorting

```
L1 = [5, 4, 2, 3, 1]
L1.sort()
print(L1)      # [1, 2, 3, 4, 5]
```

```
Ls1 = ['cat', 'mouse', 'pig', 'dog', 'bird']
Ls1.sort()
print(Ls1)     # ['bird', 'cat', 'dog', 'mouse', 'pig']
```

sorting

- `sort` method
  - Descending sorting

```
L1 = [5, 4, 2, 3, 1]
L1.sort(reverse = True)
print(L1)      # [5, 4, 3, 2 ,1]
```

```
Ls1 = ['cat', 'mouse', 'pig', 'dog', 'bird']
Ls1.sort(reverse = True)
print(Ls1)     # ['pig', 'mouse', 'dog', 'cat', 'bird']
```

sorting

- Let's try it
  - Input a text s
  - Change this text to a list of characters , L
  - Sort L by order of ASCII
  - For example
    - `s = 'I am a smart guy!'`
    - L will be:
      - `[' ', ' ', ' ', ' ', '!', 'I', 'a', 'a', 'a', 'g', 'm', 'm', 'r', 's', 't', 'u', 'y']`

sorting

- Let's try it
  - Input a positive integer n
  - Split all digits to a list of integers, `L`
  - Sort `L` by order of numbers
  - Then, transform L to an integer m
  - Print n + m
    - For example
      - `n = 8573`
      - `L` will be:
        - `[3, 5, 7, 8]`
      - then, `m = 3578`
      - `n + m = 12151`

Exercise

- Input a set of scores
  - Finding the **maximum**, **minimum** and **median**
    - The index of median in N sorted numbers is `int(N/2)`
  - Calculating the **standard deviation of median** by the following equation

$$s_m = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}(A[i]-\boldsymbol{m})^2}$$

    - where *N* is number of scores, *A* is the list of scores, and *m* is the median.