SWCON104
Web & Python Programming

Loop

Department of Software Convergence

경희대학교
KYUNG HEE UNIVERSITY

Today

- Loops!
- for-loop
- while-loop
- break and continue

[Textbook]
Practical Programming
(An Introduction to Computer Science Using Python),
by Paul Gries, Jennifer Campbell, Jason Montojo,
The Pragmatic Bookshelf, 2017

Practice

- Practice_11_Loop_Part_1.ipynb
- Practice_11_Loop_Part_2.ipynb

Repetition

- Fundamental kind of control flow
  - Making choices: `if`
  - Repeating code: `for`/`while`

- How to use loops to repeat the certain code the desired number of times.

Processing items in a list

- Editor (without loop)

```
velocities = [0.0, 9.81, 19.62, 29.43]

print('Metric:', velocities[0], 'm/sec;', 'Imperial:', velocities[0] * 3.28, 'ft/sec')
print('Metric:', velocities[1], 'm/sec;', 'Imperial:', velocities[1] * 3.28, 'ft/sec')
print('Metric:', velocities[2], 'm/sec;', 'Imperial:', velocities[2] * 3.28, 'ft/sec')
print('Metric:', velocities[3], 'm/sec;', 'Imperial:', velocities[3] * 3.28, 'ft/sec')
```

- Shell

```
Metric: 0.0 m/sec; Imperial: 0.0 ft/sec
Metric: 9.81 m/sec; Imperial: 32.1768 ft/sec
Metric: 19.62 m/sec; Imperial: 64.3536 ft/sec
Metric: 29.43 m/sec; Imperial: 96.5304 ft/sec
>>>
```

- Editor (using loop)

```
for v in velocities:
    print('Metric:', v, 'm/sec;', 'Imperial:', v * 3.28, 'ft/sec')
```

for loop

```
velocities = [0.0, 9.81, 19.62, 29.43]
for v in velocities:
    print('Metric:', v, 'm/sec;', 'Imperial:', v * 3.28, 'ft/sec')
```

for <<variable>> in <<list>>:

    <<block>>

- The loop variable is assigned the first item in the list, and the loop block (the body of the for loop) is executed.
- The loop variables is then assigned the second item in the list and the loop body is executed again.
- ……
- Finally, the loop variable is assigned the last item in the list and the loop body is executed one last time.

Iteration

- Each pass through the block is called an iteration

```
velocities = [0.0, 9.81, 19.62, 29.43]
for v in velocities:
    print('Metric:', v, 'm/sec;', 'Imperial:', v * 3.28, 'ft/sec')
```

Iteration	List Item Referred to at Start of Iteration	What Is Printed During This Iteration
1st	velocities[0]	Metric: 0.0 m/sec; Imperial: 0.0 ft/sec
2nd	velocities[1]	Metric: 9.81 m/sec; Imperial: 32.1768 ft/sec
3rd	velocities[2]	Metric: 19.62 m/sec; Imperial: 64.3536 ft/sec
4th	velocities[3]	Metric: 29.43 m/sec; Imperial: 96.5304 ft/sec

Table 11—Looping Over List Velocities

Iteration

- We can use the existing variable.
- The loop still starts with the variable referring to the first element of the list.
- The code inside the loop must be indented.
- When the loop finishes, the variable is holding its last value.

```
velocities = [0.0, 9.81, 19.62, 29.43]
speed = 2
for speed in velocities:
    print('Metric:', speed, 'm/sec')
print('Final:', speed, 'm/sec')
```

```
Metric: 0.0 m/sec
Metric: 9.81 m/sec
Metric: 19.62 m/sec
Metric: 29.43 m/sec
Final: 29.43 m/sec
>>>
```

Processing characters in strings

- `ch` is assigned `country[0]` just before the first iteration
- `ch` is assigned `country[1]` just before the second iteration
- The loop iterates 24 times
- if-statement block is executed three times

```
country = 'United States of America'
for ch in country:
    if ch.isupper():
        print(ch)
```

```
U
S
A
>>>
```

Looping over a range of numbers

- Range is a type

```
>>> a = range(10)
>>> a
range(0, 10)
>>> list(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Loop over a range

```
>>> for num in range(10):
        print(num)


0
1
2
3
4
5
6
7
8
9
```

- Loop over a list

```
velocities = [0.0, 9.81, 19.62, 29.43]
speed = 2
for speed in velocities:
    print('Metric:', speed, 'm/sec')
print('Final:', speed, 'm/sec')
```

Range

```
>>> list(range(3))
[0, 1, 2]
>>> list(range(1))
[0]
>>> list(range(0))
[]
>>> list(range(1,5))
[1, 2, 3, 4]
>>> list(range(1,10))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1,10,2))
[1, 3, 5, 7, 9]
>>> list(range(0,10,2))
[0, 2, 4, 6, 8]
>>> list(range(0,10,1))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(0,10,-1))
[]
>>> list(range(10,0,-1))
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
>>> total = 0
>>> for i in range(1,10):
        total = total + i


>>> total
45
```

```
>>> total = 0
>>> for i in range(1,11):
        total = total + i


>>> total
55
```

Processing lists using indices

- How to change the values in a list? Suppose you want to double the values.

```
values = [4, 10, 3, 8, -6]
for num in values:
    num = num * 2
    print(num)

print(values)
```

```
8
20
6
16
-12
[4, 10, 3, 8, -6]
```

```
values = [4, 10, 3, 8, -6]
for i in range(len(values)):
    print(i,values[i])
```

```
0 4
1 10
2 3
3 8
4 -6
```

```
values = [4, 10, 3, 8, -6]
for i in range(len(values)):
    values[i] = values[i] * 2

print(values)
```

```
[8, 20, 6, 16, -12]
```

Processing parallel lists using indices

- Sometimes the data from one list corresponds to data from another.

Editor

```
metals = ['Li','Na','K']
weights = [6.941, 22.98976928, 39.0983]

for i in range(len(metals)):
    print(metals[i], weights[i])
```

Shell

```
Li 6.941
Na 22.98976928
K 39.0983
```

Nested loops

- A loop can contain another loop

```
positive = ['Li','Na','K']
negative = ['F','Cl','Br']

for metal in positive:
    for halogen in negative:
        print(metal+halogen)
```

```
LiF
LiCl
LiBr
NaF
NaCl
NaBr
KF
KCl
KBr
```

Iteration of Outer Loop	What metal Refers To	Iteration of Inner Loop	What halogen Refers To	What Is Printed
1st	outer[0]	1st	inner[0]	LiF
		2nd	inner[1]	LiCl
		3rd	inner[2]	LiBr
2nd	outer[1]	1st	inner[0]	NaF
		2nd	inner[1]	NaCl
		3rd	inner[2]	NaBr
3rd	outer[2]	1st	inner[0]	KF
		2nd	inner[1]	KCl
		3rd	inner[2]	KBr

Table 12—Nested Loops Over Inner and Outer Lists

- print() is called 9 times because len(positive)*len(negative) = 9
- For each iteration of the outer loop (for each item in positive), the inner loop executes three times (once per item in negative)

Exercise: Print multiplication table

```
def print_table(n):
    """ (int) -> NoneType

    Print the multiplication table for numbers 1 through n inclusive.

    >>> print_table(5)
        1       2       3       4       5
    1   1       2       3       4       5
    2   2       4       6       8       10
    3   3       6       9       12      15
    4   4       8       12      16      20
    5   5       10      15      20      25
    """
    # The numbers to include in the table.
    numbers = list(range(1, n + 1))

    # Print the header row.
    for i in numbers:
        print('\t' + str(i), end='')

    # End the header row.
    print()

    # Print each row number and the contents of each row.
    for i in numbers:

        print (i, end='')
        for j in numbers:
            print('\t' + str(i * j), end='')

        # End the current row.
        print()
```

```
>>> print_table(4)
        1       2       3       4
1       1       2       3       4
2       2       4       6       8
3       3       6       9       12
4       4       8       12      16
>>> print_table(5)
        1       2       3       4       5
1       1       2       3       4       5
2       2       4       6       8       10
3       3       6       9       12      15
4       4       8       12      16      20
5       5       10      15      20      25
>>> print_table(6)
        1       2       3       4       5       6
1       1       2       3       4       5       6
2       2       4       6       8       10      12
3       3       6       9       12      15      18
4       4       8       12      16      20      24
5       5       10      15      20      25      30
6       6       12      18      24      30      36
```

Looping over nested lists

- Editor

```
elements = [['Li','Na','K'],['F','Cl','Br']]
for inner_list in elements:
    print(inner_list)
```

- Shell

```
['Li', 'Na', 'K']
['F', 'Cl', 'Br']
>>>
```

```
elements = [['Li','Na','K'],['F','Cl','Br']]
for inner_list in elements:
    for item in inner_list:
        print(item)
```

```
Li
Na
K
F
Cl
Br
>>>
```

Looping over ragged lists

- Ragged lists: Nested lists with inner lists of varying lengths

```
>>> drinking_times_by_day = [["9:02", "10:17", "13:52", "18:23", "21:31"],
...                          ["8:45", "12:44", "14:52", "22:17"],
...                          ["8:55", "11:11", "12:34", "13:46",
...                           "15:52", "17:08", "21:15"],
...                          ["9:15", "11:44", "16:28"],
...                          ["10:01", "13:33", "16:45", "19:00"],
...                          ["9:34", "11:16", "15:52", "20:37"],
...                          ["9:01", "12:24", "18:51", "23:13"]]

>>> for day in drinking_times_by_day:
...     for drinking_time in day:
...         print(drinking_time, end=' ')
...     print()
...
9:02 10:17 13:52 18:23 21:31
8:45 12:44 14:52 22:17
8:55 11:11 12:34 13:46 15:52 17:08 21:15
9:15 11:44 16:28
10:01 13:33 16:45 19:00
9:34 11:16 15:52 20:37
9:01 12:24 18:51 23:13
```

Looping until a condition is reached

- for loops
  - You know how many iterations of the loop you need

```
for <<variable>> in <<list>>:
    <<block>>
```

- while loops
  - It is not known in advance how many loop iterations to execute

```
while <<expression>>:
    <<block>>
```

Loop condition: just like the condition of an if-statement

```
if <<condition>>:
    <<block>>
```

while loop

```
while <<expression>>:
    <<block>>
```

- When Python executes a `while` loop,
  - Python evaluates the expression.
  - If the expression evaluates to `False`, that is the end of the execution of the loop.
  - If the expression evaluates to `True`, Python executes the loop body once and then goes back to the top of the loop and reevaluates the expression.

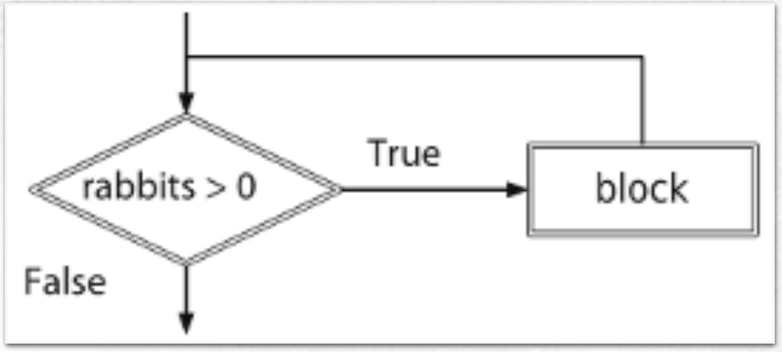
While-loop example 1

- Editor

```
rabbits = 3
while rabbits > 0:
    print(rabbits)
    rabbits = rabbits - 1
```

- Shell

```
3
2
1
>>>
```



- When Python executes a `while` loop,
  - Python evaluates the expression.
  - If the expression evaluates to `False`, that is the end of the execution of the loop.
  - If the expression evaluates to `True`, Python executes the loop body once and then goes back to the top of the loop and reevaluates the expression.

While-loop example 2

- Growth of bacterial population

```
time = 0
population = 1000 # 1000 bacteria at the start
growth_rate = 0.21 # 21% growth per minute

while population < 2000:
    population = population + growth_rate * population
    print(round(population))
    time = time + 1

print("It took", time, "minutes for the bacteria to double.")
print("The final population was", round(population), "bacteria.")
```

```
1210
1464
1772
2144
It took 4 minutes for the bacteria to double.
The final population was 2144 bacteria.
```

- Exponential growth model
  - How long does it take the bacteria to double their numbers?
  - For example, if the population was 1000 at the start, how long does it take to be 2000, when the growth rate is given.

Infinite loops

- If we set `while`-loop condition to be always false…

```
time = 0
population = 1000 # 1000 bacteria at the start
growth_rate = 0.21 # 21% growth per minute

while population == 2000:
    population = population + growth_rate * population
    print(round(population))
    time = time + 1

print("It took", time, "minutes for the bacteria to double.")
print("The final population was", round(population), "bacteria.")
```

```
It took 0 minutes for the bacteria to double.
The final population was 1000 bacteria.
```

- If we set `while`-loop condition to be always true…

```
time = 0
population = 1000 # 1000 bacteria at the start
growth_rate = 0.21 # 21% growth per minute

while population != 2000:
    population = population + growth_rate * population
    print(round(population))
    time = time + 1

print("It took", time, "minutes for the bacteria to double.")
print("The final population was", round(population), "bacteria.")
```

INFINITE LOOP

Ctrl+C to stop the program

Repetition based on user input

- An interactive program
  - The user enters a chemical formula, the program answers
  - until the user enters the command to quit the program
  - The number of times that this loop executes will vary depending on user input, but it will execute at least once.

```
text = ""
while text != "quit":
    text = input("Please enter a chemical formula (or 'quit' to exit): ")
    if text == "quit":
        print("…exiting program")
    elif text == "H2O":
        print("Water")
    elif text == "NH3":
        print("Ammonia")
    elif text == "CH4":
        print("Methane")
    else:
        print("Unknown compound")
```

```
Please enter a chemical formula (or 'quit' to exit): H2O
Water
Please enter a chemical formula (or 'quit' to exit): CH4
Methane
Please enter a chemical formula (or 'quit' to exit): NH3
Ammonia
Please enter a chemical formula (or 'quit' to exit): NH3
Ammonia
Please enter a chemical formula (or 'quit' to exit): quit
…exiting program
```

Controlling loops using break and continue

- break terminates execution of the loop

```
text = ""
while True:
    text = input("Please enter a chemical formula (or 'quit' to exit): ")
    if text == "quit":
        print("…exiting program")
        break
    elif text == "H2O":
        print("Water")
    elif text == "NH3":
        print("Ammonia")
    elif text == "CH4":
        print("Methane")
    else:
        print("Unknown compound")
```

```
while(for) <<expression>>
    …
    if <<condition>>
        break
    …
```

break example

- Editor

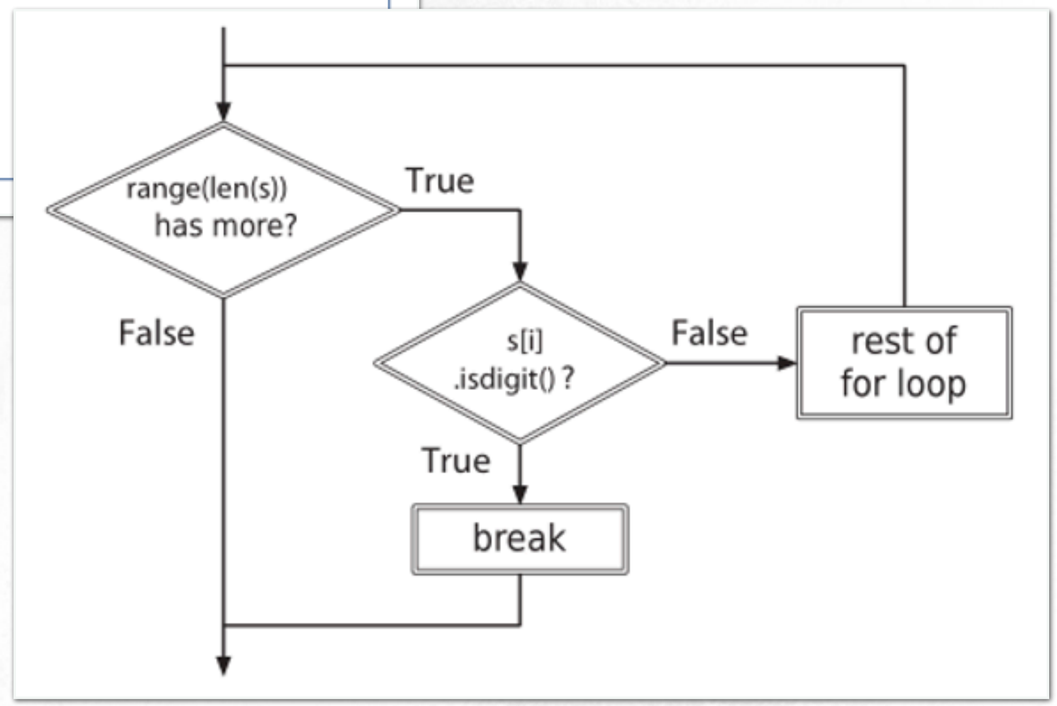
```
s = 'C3H7'
digit_index = -1 # This will be -1 until we find a digit.
for i in range(len(s)):
    # If we find a digit
    if s[i].isdigit():
        digit_index = i
        print(digit_index)
```

```
s = 'C3H7'
digit_index = -1 # This will be -1 until we find a digit.
for i in range(len(s)):
    # If we find a digit
    if s[i].isdigit():
        digit_index = i
        print(digit_index)
        break
```

- Shell

```
1
3
>>>
```

```
1
>>>
```



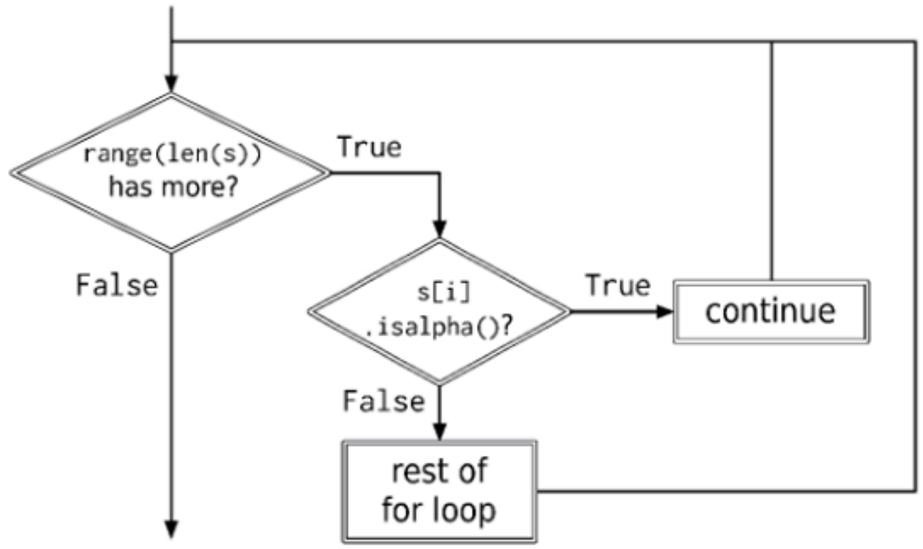
continue statement

- continue: skip immediately to the next iteration of the loop

```
s = 'C3H7'
total = 0 # The sum of the digits
count = 0 # The number of digits
for i in range(len(s)):
    if s[i].isalpha():
        continue
    total = total + int(s[i])
    count = count + 1

print('total:',total)
print('count:',count)
```

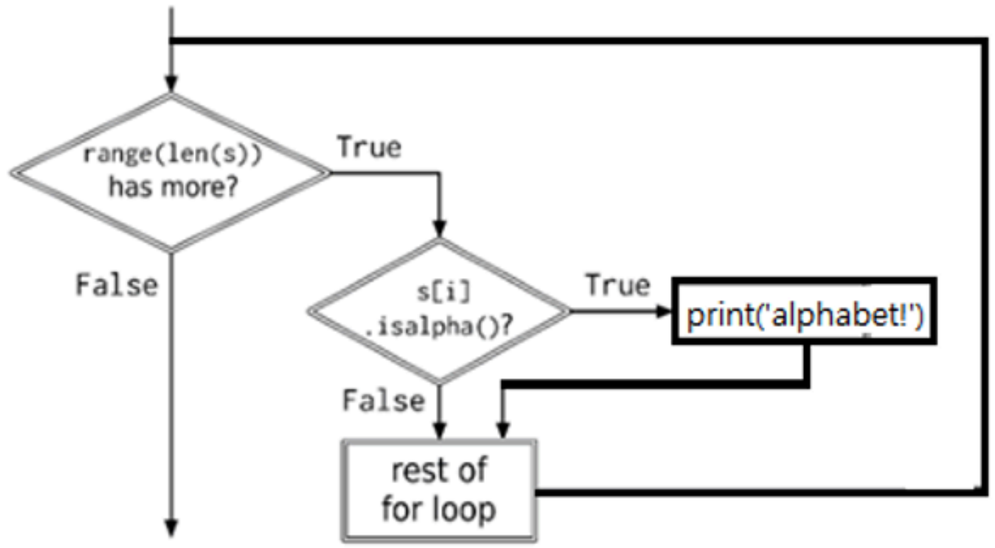
```
total: 10
count: 2
>>>
```




```
s = 'C3H7'
total = 0 # The sum of the digits
count = 0 # The number of digits
for i in range(len(s)):
    if s[i].isalpha():
        print('alphabet!')
    #total = total + int(s[i])
    count = count + 1

print('total:',total)
print('count:',count)
```

```
alphabet!
alphabet!
total: 0
count: 4
>>>
```



Warning about `break` and `continue`

- They tend to make programs harder to understand.
- There are always alternatives:
  - Well-chosen loop conditions can replace `break`
  - `if`-statements can be used to skip statements instead of `continue`
- It is up to the programmer to decide which option makes the program clearer and which makes it more complicated

Summary

- Repeating a block is a fundamental way to control a program's behavior. A `for` loop can be used to iterate over the items of a list, over the characters of a string, and over a sequence of integers generated by built-in function `range`.
- The most general kind of repetition is the `while` loop, which continues executing as long as some specified Boolean condition is true. However, the condition is tested only at the beginning of each iteration. If that condition is never false, the loop will be executed forever.
- The `break` and `continue` statements can be used to change the way loops execute.
- Control structures like loops and conditionals can be nested inside one another to any desired depth.

Thank you

경희대학교
KYUNG HEE UNIVERSITY