AI전문가과정 - Python Review

# Repetition Structures

Jinwook Seo, Ph.D.
Professor, Department of Computer Science and Engineering
Seoul National University

# Table of Contents

# Repetion Structures

- Often have to write code that performs the same task multiple times
  - Disadvantages of duplicating code
    - Time consuming to type in code
    - Makes program large
    - May need to be corrected in many places
- Repetition structure makes computer repeat included code as necessary
  - condition-controlled loops (`while`)
  - count-controlled loops (`for`)

AI전문가과정 - Python Review

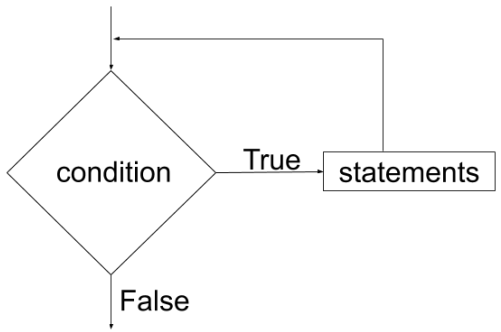
# `while` Loop

# `while` Loop

- condition-controlled loop

```
while condition:
    statements
```

- While `condition` is true, do something
  - condition tested for `True` or `False` value
  - statements repeated as long as condition is `True`
  - In flowchart, line goes back to previous part



# `while` Loop

- In order for a loop to stop executing, something has to happen inside the loop to make the condition false
- Iteration: one execution of the body of a loop
- `while` loop is known as a pretest loop
  - Tests `condition` before performing an iteration
  - Will never execute if `condition` is false to start with
  - Requires performing some steps prior to the loop

# while Loop

```
keep_going = 'y'

# Calculate a series of commissions.
while keep_going == 'y':
    # Get a salesperson's sales and commission rate.
    sales = float(input('Enter the amount of sales: '))
    comm_rate = float(input('Enter the commission rate: '))

    # Calculate the commission.
    commission = sales * comm_rate

    # Display the commission.
    print('The commission is $', \
    format(commission, ',.2f'), sep='')

    # See if the user wants to do another one.
    keep_going = input('Do you want to calculate another ' + \
                       'commission (Enter y for yes): ')
```

Run Code | Visualize

# Infinite Loops

- Loops must contain within themselves a way to terminate
  - Some statement inside a while loop must eventually make the condition `false`
- Infinite loop: loop that does not have a way of stopping
  - Repeats until program is interrupted
  - Occurs when programmer forgets to include appropriate stopping code in the loop

```
count = 1;
while count > 0:
    print("iteration count = ", count);
    count = count + 1
```

# `while` Loop Example

- Using exhaustive enumeration to find the cube root

```
# find the cube root of a perfect cube

x = int(input("Enter an integer: "))
ans = 0
while ans**3 < abs(x):
    ans = ans + 1

if ans**3 != abs(x):
    print(x, 'is not a perfect cube.')
else:
    if x < 0:
        ans = -ans
    print('Cube root of', x, 'is', ans, '.')
```

Run Code Visualize

AI전문가과정 - Python Review

# for Loop

# for Loop

- count-controlled loop iterates a specific number of times
- Use a `for` statement to write a count-controlled loop
  - Designed to work with a sequence of data items (e.g., a `list`)
  - Iterates once for each item in the sequence

```
for variable in [val1, val2, etc]:
    statements
```

```
                      ┌─────────────┐
1st iteration:   for num in [1, 2, 3]:
                     print(num)

                      ┌──────────────────┐
2nd iteration:   for num in [1, 2, 3]:
                     print(num)

                      ┌─────────────────────┐
3rd iteration:   for num in [1, 2, 3]:
                     print(num)
```

# for Loop

- count-controlled loop iterates a specific number of times

```
>>> for name in ['John', 'James', 'Jane']:
...     print(name)
...
John
James
Jane
```

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']  # create a list object
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

# `range` Class with `for` Loops

- The `range` class simplifies the process of writing a `for` loop
- `range` type object is an iterable object
  - Iterable object contains a sequence of values that can be iterated over
  - It is suitable as a target for functions and constructs that expect something from which they can obtain successive items until the supply is exhausted.

```
class range(stop)
class range(start, stop[, step])
```

- If the `start` argument is omitted, it defaults to `0`.
- `stop` is not included in the sequence
- If the `step` argument is omitted, it defaults to `1`.
- For a positive `step`, the contents of a `range r` are determined by the formula `r[i] = start + step*i` where `i >= 0` and `r[i] < stop`.
- For a negative `step`, the contents of the `range` are still determined by the formula `r[i] = start + step*i`, but the constraints are `i >= 0` and `r[i] > stop`.

# `range` Class with `for` Loops

```
class range(stop)
class range(start, stop[, step])
```

- `stop` is not included in the sequence
- The followings are all equivalent!

```
for i in range(0, 5):
for i in range(5):
for i in [0, 1, 2, 3, 4]:
```

# range Class with for Loops

```
for num in [1, 2, 3, 4, 5]:   # iterate over a "list"
    print(num, end=' ')
for num in range(1, 6):
    print(num, end=' ')
# prints 1 2 3 4 5

for num in range(5):
    print(num, end=' ')
# prints 0 1 2 3 4

for num in range(1, 10, 2):
    print(num, end=' ')
# prints 1 3 5 7 9
```

# Iterating Over a String

```
for num in range(0, -10, -2):
    print(num, end=' ')
# prints 0 -2 -4 -6 -8

for num in range(0):
    print(num)
# prints nothing

for c in "I'm learning Python":
    print(c)
```

Run Code | Visualize

# for Loop Example

```
# Print the table headings.
print('Number\tSquare')
print('--------------')

# Print the numbers 1 through 10
# and their squares.

for number in range(1, 11):
    square = number**2
    print(number, '\t', square)
```

Run Code Visualize

# for Loop Example

```
print('This program displays a list of numbers')
print('(starting at 1) and their squares.')
end = int(input('How high should I go? '))

# Print the table headings.
print('Number\tSquare')
print('--------------')

# Print the numbers 1 through 10
# and their squares.

for number in range(1, end + 1):  # range doesn't include stop
    square = number**2
    print(number, '\t', square)
```

Run Code Visualize

# for Loop Example

```
n = int(input("Please enter an integer: "))
result = 1

for factor in range(n, 1, -1):
    result = result * factor

print("The factorial of", n, "is", result)
```

Run Code Visualize

# for Loop Example

- Is it possible to change the end condition inside the loop?

```
x = 4
for i in range(0, x):
    print(i)
    x=5
```

Run Code Visualize

- The argument x is evaluated just once before the first iteration!

# for Loop Example

- A Few Words About Using Floating-point Numbers

```
x = 0.0

for i in range(10):
    x = x + 0.1    # 0.1 = 0.0001100110011001100110011.... in binary representation

if x == 1.0:
    print(x, '= 1.0')
else:
    print(x, 'is not 1.0')
```

Run Code Visualize

# for Loop Example

- A Few Words About Using Floating-point Numbers

```
epsilon = 0.001
if abs(x - 1.0) < epsilon:
# do something
```

- Chopping errors could make devastating results
  - Software Problem Led to System Failure at Dhahran, Saudi Arabia
    - https://www.gao.gov/assets/220/215614.pdf
    - http://www-users.math.umn.edu/~arnold/disasters/patriot.html

```
"Because of the way the Patriot computer performs its calculations and
the fact that its registers are only 24 bits long,
the conversion of time from an integer to a real number
cannot be any more precise than 24 bits.
This conversion results in a loss of precision
causing a less accurate time calculation."
```

# Calculating a Running Total

- Programs often need to calculate a total of a series of numbers
  - Typically include two elements:
    - A loop that reads each number in series
    - An accumulator variable
  - Known as program that keeps a running total: accumulates total and reads in series
  - At end of loop, accumulator will reference the total

```
# Constant for the maximum number
MAX = 5

# Initialize an accumulator variable.
total = 0.0

# Explain what we are doing.
print('This program calculates the sum of')
print(MAX, 'numbers you will enter.')

# Get the numbers and accumulate them.
for counter in range(MAX):
    number = int(input('Enter a number: '))
    total = total + number

# Display the total of the numbers.
print('The total is', total)
```

```
>>>
This program calculates the sum of
5 numbers you will enter.
Enter a number: 10
Enter a number: 20
Enter a number: 30
Enter a number: 40
Enter a number: 50
The total is 150.0
>>>
```

AI전문가과정 - Python Review

# `break` and `continue`

hci lab
SEOUL NATIONAL UNIVERSITY

# Using for and break statements

```
# find the cube root of a perfect cube

x = int(input("Enter an integer: "))

for ans in range(0, abs(x) + 1):
    if ans**3 >= abs(x):
        break

if ans**3 != abs(x):
    print(x, 'is not a perfect cube.')
else:
    if x < 0:
        ans = -ans
    print('Cube root of', x, \
          'is', ans, '.')
```

Run Code Visualize

```
# find the cube root of a perfect cube

x = int(input("Enter an integer: "))
ans = 0
while ans**3 < abs(x):
    ans = ans + 1

if ans**3 != abs(x):
    print(x, 'is not a perfect cube.')
else:
    if x < 0:
        ans = -ans
    print('Cube root of', x, \
          'is', ans, '.')
```

Run Code Visualize

hci lab
SEOUL NATIONAL UNIVERSITY

# Using for and continue statements

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print("Found an even number", num)
...         continue
...     print("Found a number", num)
Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9
```

```
for i in range(30):
    if not (i%3) :
        print("*")
        continue
    elif str(i).find('3') != -1:
        print("**")
        continue
    print(i)
```

Run Code | Visualize

# `break` and `else` Clause on Loops

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

# `break` and `else` Clause on Loops

- `else` clause belongs to the `for` loop, not the `if` statement
- `else` clause is executed when the loop terminates
  - through exhaustion of the iterable (with `for`) or
  - when the condition becomes `false` (with `while`)
  - but not when the loop is terminated by a `break` statement.

AI전문가과정 - Python Review

# Sentinal & Nested Loop

# Augmented Assignment Statement

| operator | example | equivalent to |
|---|---|---|
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| *= | x *= 3 | x = x * 3 |
| /= | x /= 3 | x = x / 3 |
| //= | x //= 3 | x = x // 3 |
| %= | x %= 3 | x = x % 3 |
| **= | x **= 3 | x = x ** 3 |
| @= | | |
| >>= | | |
| <<= | | |
| &= | | |
| ^= | | |
| \|= | | |

- Unlike normal assignments, augmented assignments evaluate the left-hand side first before evaluating the right-hand side.
- For example, `a[i] += f(x)`
  - first looks-up `a[i]`,
  - then it evaluates `f(x)`
  - and performs the addition,
  - and lastly, it writes the result back to `a[i]`.

# Sentinel

- Sentinel: special value that marks the end of a sequence of items

```
TAX_FACTOR = 0.0065

# Get the first lot number.
print('Enter the property lot number')
print('or enter 0 to end.')
lot = int(input('Lot number: '))

while lot != 0:

    value = float(input('Enter the property value: '))

    tax = value * TAX_FACTOR

    print('Property tax: $', format(tax, ',.2f'), sep='')

    # Get the next lot number.
    print('Enter the next lot number or')
    print('enter 0 to end.')
    lot = int(input('Lot number: '))
```

# Input Validation Loop

- Computer cannot tell the difference between good data and bad data
- If user provides bad input, program will produce bad output
- GIGO: garbage in, garbage out
- It is important to design program such that bad input is never accepted
- Error Prevention is much better than Error Handling

```
# Get a test score.
score = int(input('Enter a test score: '))
# Make sure it is not less than 0 or greater than 100.
while score < 0 or score > 100:
    print('ERROR: The score cannot be negative')
    print('or greater than 100.')
    score = int(input('Enter the correct score: '))
# do something with score
```

# Nested Loop

- Nested loop: a loop can be contained inside another loop

```
for hours in range(24):
    for minutes in range(60):
        for seconds in range(60):
            print(hours, ':', minutes, ':', seconds)
```

- Is it possible to change the end condition inside the loop?
- The argument `x` is evaluated just before the first iteration!

```
x = 4
for j in range(x):
    for i in range(x):
        print(i)
        x = 2
```

Run Code | Visualize

# Wrap-UP

- A `while` statement is used to iterate continuously until a condition is true.
- A `for` loop is used for iterating over a sequence.
- A `break` statement is used to break out of a loop.
- A `continue` statement is used to skip a specific part of the loop and continue executing the loop.
- A nested loop is a loop inside a loop.

AI전문가과정 - Python Review

# Q&A

# Acknowledgement


- The Python Tutorial, https://docs.python.org/3/tutorial/index.html
- Lecture Notes, Professor Hyungjoo Kim
- Starting out with Python, Professor Tony Gaddis and Pearson Education, Ltd.
- Introduction to Computation and Programming Using Python, John V. Guttag