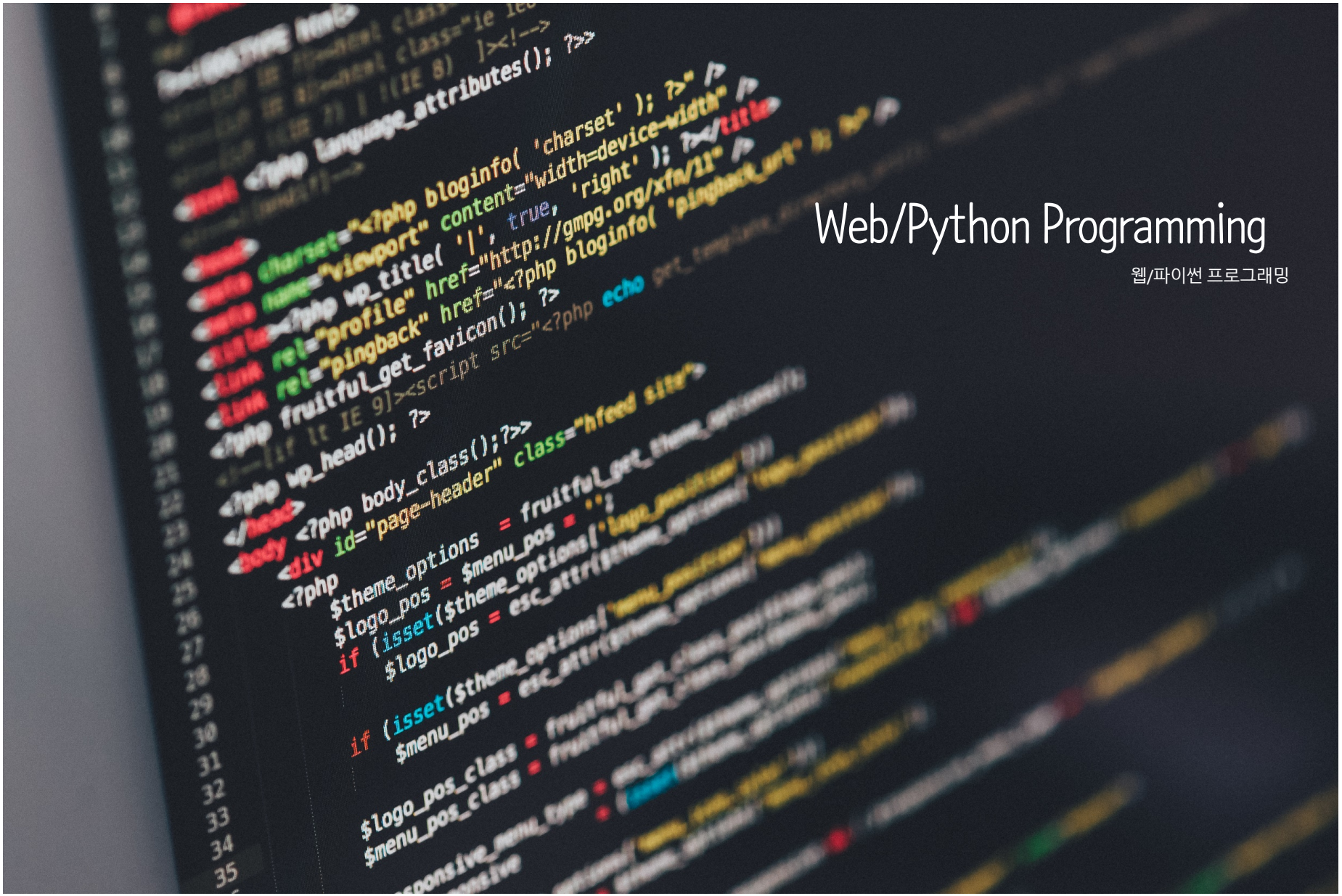


Web/Python Programming

웹/파이썬 프로그래밍

Today

- Review `for` loop and `while` loop
- `break` and `continue`
- File I/O

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.

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]
```

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.

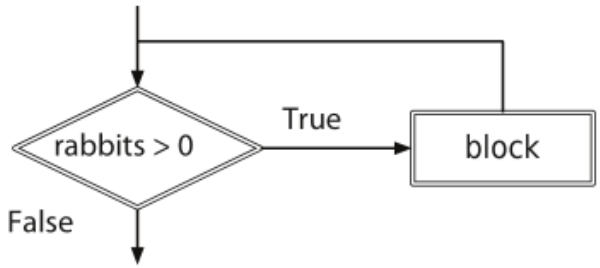
While-loop example 1

- 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.

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 th
print("The final population was", roun
```

```
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

```
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` terminates execution of the loop

break example

- Editor

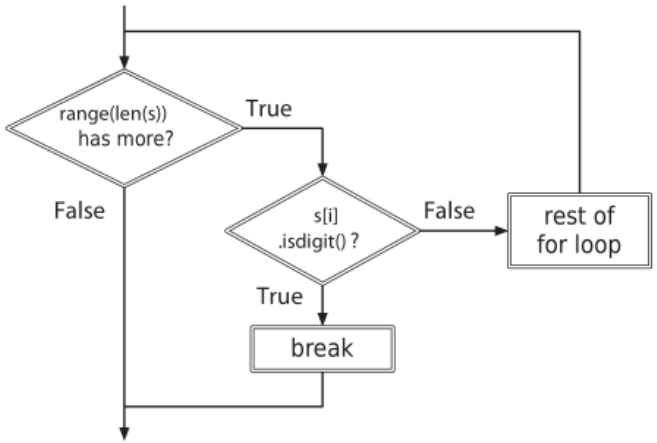
```
s = 'C3H7'
digit_index = -1 # This will be -1 until we find a digit.
for i in range(len(s)):
    # If we find a digit
    if s[i].isdigit():
        digit_index = i
        print(digit_index)
```

- Shell

```
1
3
>>> |
```

```
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
```

```
1
>>> |
```



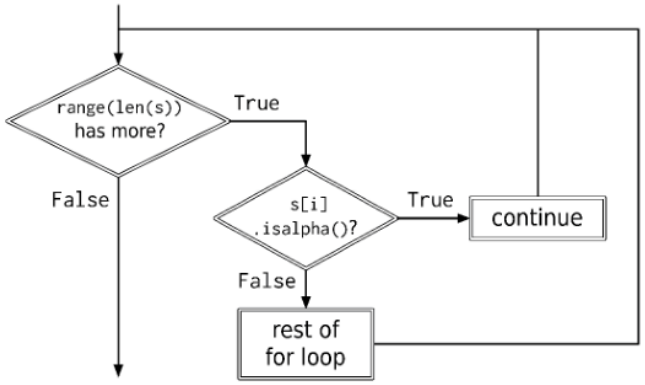
continue statement

- `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)
```

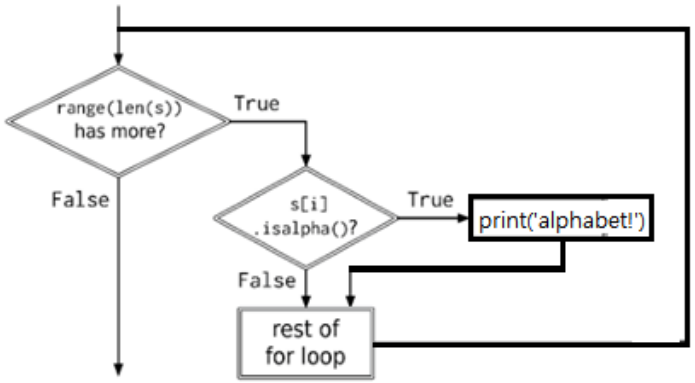
```
total: 10
count: 2
>>>
```




```
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.

How to get data?

```
if age < 45:
    if bmi < 22.0:
        risk = 'low'
    else:
        risk = 'medium'
else:
    if bmi < 22.0:
        risk = 'medium'
    else:
        risk = 'high'
```

$$\text{Body Mass Index (bmi)} = \frac{\text{weight (kg)}}{\left(\text{height (m)}\right)^2}$$

Name: Mike
Age: 24
Height: 172
Weight: 72

Name: Jane
Age: 51
Height: 160
Weight: 60

Name: Jason

Mike, 24, 172, 72
Jane, 51, 160, 60
Jason, 35, 180, 80
…

Mike, Jane, Jason;
24, 51, 35;
172, 160, 180;
72, 60, 80;
…

Mike 24 172 72; Jane 51 160 60; Jason 35 180 80;
…

What kind of files are there?

- Text files
- Music files
- Videos
- Word processor (docx, hwp)
- Presentation documents (ppt)
- Spread sheets (excel)
- pdf

- Text files only contain characters
- Other file formats include formatting information that is specific to that particular file format
- Ex) You cannot open a ppt file using notepad
- Ex) Check the size of an empty file of various format

Text files

- Take up little disk space
- Easy to process
- Only letters in a file

- .py file is a text file
- With a particular syntax
- Python interpreter can read Python text files and follow the instructions

- Web browsers read and process HTML files
- Spreadsheets read and process comma-separated value files
- Calendar programs read and process calendar data files
- Other programming language applications read and process files written with a particular programming language syntax

Opening a file

- Python assumes that the file you want to read is in the same directory as the current program

1) Make a directory, `file_examples`
2) Open Notepad and type the following:

```
First line of text
Second line of text
Third line of text
```

3) Save this file in your `file_examples` directory as `file_example.txt`
4) In IDLE, select File-> New Window and type this program:
5) Save this as `file_reader.py` in `file_examples` directory
6) Run

```
file = open('file_example.txt','r')
contents = file.read()
print(contents)
file.close()
```

Opening a file

file name

file mode

```
file = open('file_example.txt','r')
contents = file.read()
print(contents)
file.close()
```

file cursor

```
First line of text
Second line of text
Third line of text
```

- Built-in function `open` opens a file and returns an object that knows…
  - How to get information from the file
  - How much you've read
  - Which part of the file you're about to read next

A file cursor is a marker that keeps track of the current location in the file.

The file cursor is initially at the beginning of the file.

As we read or write data, it moves to the end of what we just read or wrote.

The with statement

```
file = open('file_example.txt','r')
contents = file.read()
print(contents)
file.close()
```

```
with open('file_example.txt', 'r') as file:
    contents = file.read()

print(contents)
```

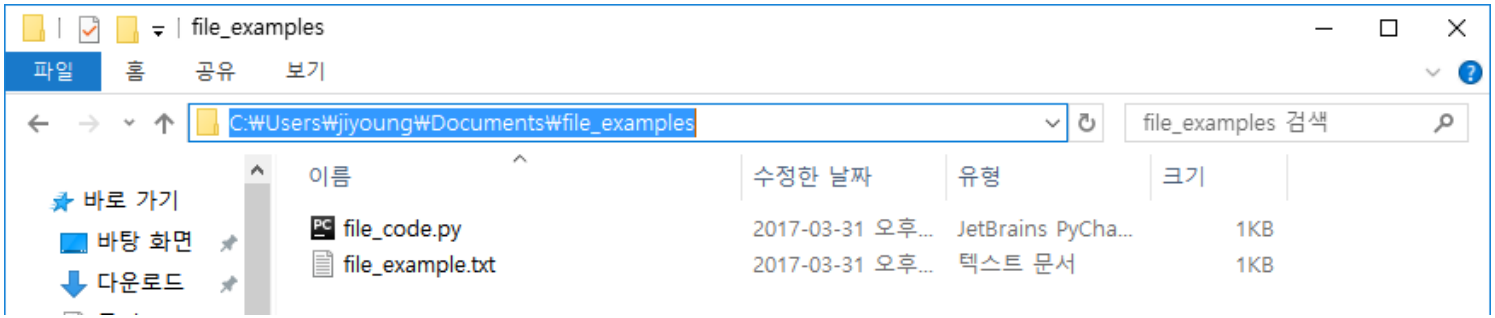
The general form of a with statement is as follows:

```
with open(«filename», «mode») as «variable»:
    «block»
```

- Automatically closes a file when the end of the block is reached
- Not recommended

How files are organized in your computer

- A file path specifies a location in your computer's file system.
- File path for file_example.txt:
  C:\Users\jiyoung\Documents\file_examples\file_example.txt



How files are organized in your computer

```
>>> path = "C:\Users\jiyoung\Documents\file_examples\file_example.txt"
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in positio
n 2-3: truncated \UXXXXXXXX escape
>>> path = "C:\\Users\\jiyoung\\Documents\\file_examples\\file_example.txt"
>>> path
'C:\\Users\\jiyoung\\Documents\\file_examples\\file_example.txt'
>>> print(path)
C:\Users\jiyoung\Documents\file_examples\file_example.txt
>>> file = open(path,'r')
>>> c = file.read()
>>> file.close()
>>> c
'first line of text\nsecond line of text\nthird line of text'
>>> path2 = "C:/Users/jiyoung/Documents/file_examples/file_example.txt"
>>> file = open(path2,'r')
>>> c2 = file.read()
>>> file.close()
>>> print(c2)
first line of text
second line of text
third line of text
```

Specifying which file you want

- Current working directory
- The directory where Python looks for files
- The directory where the current program (.py file) is saved

```
>>> import os
>>> os.getcwd()
'C:\\Users\\jiyoung\\AppData\\Local\\Programs\\Python\\Python35'
```

- An absolute path starts at the root directory of the file system

```
>>> import os
>>> os.chdir('F:\\course\\2017s_python\\scripts\\fileio')
>>> os.getcwd()
'F:\\course\\2017s_python\\scripts\\fileio'
```

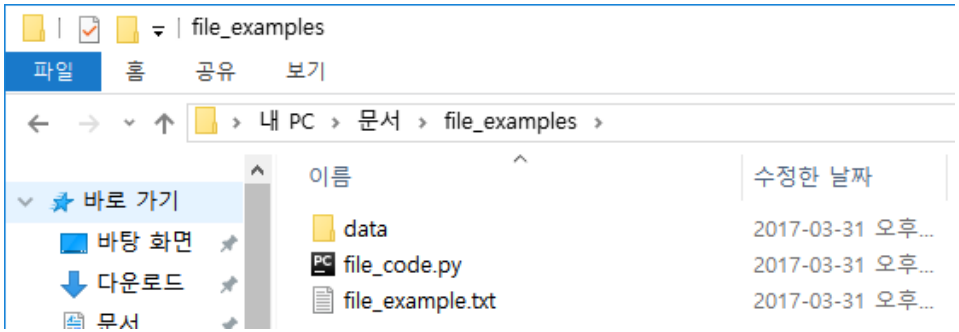
Specifying which file you want

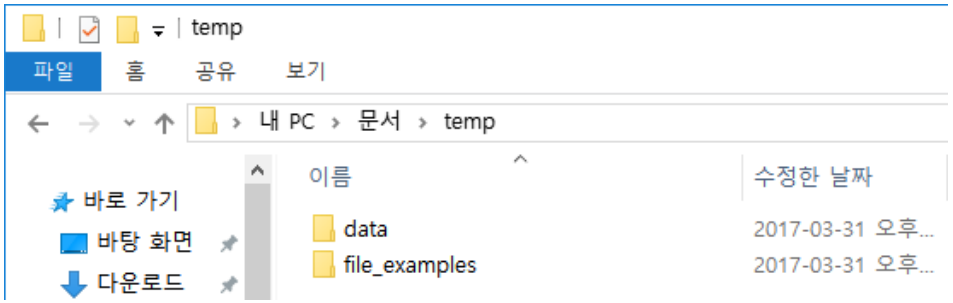
```
file = open('file_example.txt','r')
contents = file.read()
print(contents)
file.close()

file = open('data/data1.txt','r')

file = open('../data/data1.txt','r')

file = open('../../../data/data1.txt','r')
```





Relative path is relative to the current working directory