15-110 Principles of Computing – F19

Lecture 24:
Files I/O 4, String Formatting

Teacher:
Gianni A. Di Caro

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Files (and exceptions) so far

- Open a file:

```
f = open(file_name, mode)    # mode: r, w, a, x, +, t, b
```

- Read at most nbytes, or everything:

```
data_string = f.read(nbytes)
```

- Read one line record:

```
data_string = f.readline()
```

- Read all remaining line records from current position:

```
list_of_strings = f.readlines()
```

- Get current position in file:

```
pos = f.tell()
```

- Go to position pos in file:

```
f.seek(pos)
```

- Iterate through records:

```
for line in f:
    # do something with line
```

- Write a string of data:

```
f.write(data_string)
```

- Close a file:

```
f.close()
```

- Handling errors:

```
try:
    #possible error-generating code
except:
    # if an error was thrown
else:
    # if an error wasn't thrown
finally:
    # do this anyway
```

Practice

Implement the function `read_and_copy(filename)` that takes a string with a file name as input.

The function reads the file and creates another file, named `copy.txt` that contains the same records as
the input file, <u>except the last record</u>.

If the file `copy.txt` is already there, we destroy its content / overwrite it.

The function returns a tuple with the number of records and number of bytes in the input file.

If the input file doesn't exist or isn't readable, the function prints out a warning message and returns `None`.

Practice

```
def read_and_copy(filename):
    try:
        f = open(filename, 'r')
    except:
        print('Warning: file is not there!')
        return
    all_records = f.readlines()
    #print(all_records)

    fw = open('copy.txt', 'w')

    num_records = len(all_records)
    num_bytes = 0
    for r in all_records:
        num_bytes += len(r)

    for i in range(num_records-1):
        fw.write(all_records[i])

    return (num_records, num_bytes)
```

```
print(read_and_copy('data.txt'))
```

Practice

```
def copy_exact(infile, outfile):
    try:
        f = open(infile, 'r')
        fw = open(outfile, 'w')
    except:
        print('Warning: problems with files!')
        return
    all_data = f.read()
    fw.write(all_data)
```

```
copy_exact('data.txt', 'copy1.txt')
```

Output / string formatting (writing well organized/readable files)

```
f = open('personal_data.txt', 'w+')

name = 'John'
title = 'Mr.'
age_y = 30
height = 500 / 2.8
weight = 86 * 0.94
record = name + ' ' + title + ' ' + str(age) + ' ' + str(height) + ' ' + str(weight)
f.write(record + '\n')

name = 'Anne-Marie'
title = 'Ms.'
age_y = 26
height = 500 / 3
weight = 59 * 0.94
record = name + ' ' + title + ' ' + str(age) + ' ' + str(height) + ' ' + str(weight)
f.write(record + '\n')

f.close()
```

```
John Mr. 30 178.57142857142858 80.83999999999999
Anne-Marie Ms. 30 166.66666666666666 55.459999999999994
```

hard to read, misaligned fields, unnecessary digits…

Output / string formatting: `str.format()`

```
f = open('personal_data.txt', 'w+')

name = 'John'
title = 'Mr.'
age_y = 30
height = 500 / 2.8
weight = 86 * 0.94
record = '{:12s} {:4s} {:3d}  {:8.3f}  {:8.3f}'.format(name, title, age_y, height, weight)
f.write(record + '\n')

name = 'Anne-Marie'
title = 'Ms.'
age_y = 26
height = 500 / 3
weight = 59 * 0.94
record = '{:12s} {:4s} {:3d}  {:8.3f}  {:8.3f}'.format(name, title, age_y, height, weight)
f.write(record + '\n')

f.close()
```

```
John         Mr.  30  178.571  80.840
Anne-Marie   Ms.  26  166.667  55.460
```

Output / string formatting: `str.format()`

```
f = open('personal_data.txt', 'w+')

name = 'John'
title = 'Mr.'
age_y = 30
height = 500 / 2.8
weight = 86 * 0.94
record ='|{:12s}| |{:4s}| |{:3d}| |{:8.3f}| |{:8.3f}|'.format(name, title,age_y,height,weight)
f.write(record + '\n')

name = 'Anne-Marie'
title = 'Ms.'
age_y = 26
height = 500 / 3
weight = 59 * 0.94
record ='|{:12s}| |{:4s}| |{:3d}| |{:8.3f}| |{:8.3f}|'.format(name,title,age_y, height,weight)
f.write(record + '\n')

f.close()
```

```
|John        | |Mr. | | 30| | 178.571| |  80.840|
|Anne-Marie  | |Ms. | | 26| | 166.667| |  55.460|
```

String fields are left-aligned
Numeric fields are right-aligned

str.format(): positional substitution, formatting specifiers

```
'{:12s} {:4s} {:3d}  {:8.3f}  {:8.3f}'.format(name, title, age_y, height, weight)
```

the string being constructed with formatting specifications fields

the variables/values to be substituted in the formatting specifications fields

left-aligned string field of 4 chars

right-aligned float field of 8 chars, 3 after .

John	Mr.	30	178.571	80.840
Anne-Marie	Ms.	26	166.667	55.460

left-aligned string field of 12 chars

right-aligned int field of 3 chars

right-aligned float field of 8 chars, 3 after .

<u>Formatting specifiers:</u>

s: for strings

d: for decimal integers

f: for floats

b: for Booleans

e/E: exponential notation

g: general notation, rounds up to 6 significant digits

str.format(): positional substitution, formatting specifiers

- **Positional assignment** of variables to formatting specifications (in this case)

Argument index:

```
   0        1        2         3          4                  0        1        2         3          4
'{:12s} {:4s} {:3d}  {:8.3f}  {:8.3f}'.format(name, title, age_y, height, weight)
```

- Type of variable and formatting specifier must match!

```
name = 'John'
title = 'Mr.'
age_y = 30
height = 500 / 2.8
weight = 86 * 0.94
record = '{:12s} {:4s} {:3d}  {:3d}  {:3d}'.format(name, title, age_y, height, weight)
```

Throws an error since `height` and `weight` are float types while the specifier is `d` (integer)

str.format(): positional substitution, without formatting

- **Formatting specifiers are <u>not required</u>**

```
name = 'John'
title = 'Mr.'
age_y = 30
height = 500 / 2.8
weight = 86 * 0.94
record = '{} {} {} {} {}'.format(name, title, age_y, height, weight)
print(record)
print(name, title, age_y, height, weight)
```

```
John Mr. 30 178.57142857142858 80.83999999999999
John Mr. 30 178.57142857142858 80.83999999999999
```

str.format(): substitution by name

- **Values can be assigned by <u>naming the formatting fields</u>**

```
name = 'John'
title = 'Mr.'
age_y = 30
height = 500 / 2.8
weight = 86 * 0.94
record = '{Name:12s} {Title:4s} {Age:3d} {Height:8.3f} {Weight:8.3f}'.format(Name=name,
                                    Title=title, Age=age_y, Height= height, Weight=weight)
print(record)
```

```
John        Mr.  30 178.571  80.840
```

<u>General form:</u>

```
'{Field_name_1:FormattingSpecifier1} {Field_name_2:FormattingSpecifier2}'.format(
                    Field_name_1 = variable/value, Field_name_2 = variable/value)
```

str.format(): mixing up positional and named substitution

- **Positional fields must be placed <u>before named ones</u>**

```
name = 'John'
title = 'Mr.'
age_y = 30
height = 500 / 2.8
weight = 86 * 0.94
record = '{} {} {:3d} {Height:g} {Weight:e}'.format(name, title, age_y,
                                                    Height= height, Weight=weight)
print(record)
```

```
John Mr.  30 178.571 8.084e+01
```

str.format(): string formatting options

- **Align left**, with padding spaces

```
name = 'John'
record = '|{:10s}|'.format(name)
print(record)
```

```
name = 'John'
record = '|{:<10s}|'.format(name)
print(record)
```

```
|John    |
```

- **Align right**, with padding spaces

```
name = 'John'
record = '|{:>10s}|'.format(name)
print(record)
```

```
|    John|
```

str.format(): string formatting options

- **Align left,** with selected padding character

```
name = 'John'
record = '|{:_<10s}|'.format(name)
print(record)
```

```
|John______|
```

- **Align right,** with selected padding character

```
name = 'John'
record = '|{:+>10s}|'.format(name)
print(record)
```

```
|++++++John|
```

str.format(): string formatting options

- **Center** the text in the field

```
name = 'John'
record = '|{:^10s}|'.format(name)
print(record)
```

```
|   John   |
```

- **Truncate** the string up to a given number of characters

```
name = 'John'
record = '|{:.2s}|'.format(name)
print(record)
```

```
|Jo|
```

Many more options for string formatting …

- The `str.format()` method offers a wide range of possibilities (much wider than presented here!)

  https://www.programiz.com/python-programming/methods/string/format

- Similar effects can be achieved using also other methods **(% strings**, and **f-strings**)

  https://realpython.com/python-string-formatting/