ISE 314X
Computer Programing for Engineers

Chapter 5
Sequences: Strings, Lists, and Files (Part II)

Yong Wang
Assistant Professor
Systems Science & Industrial Engineering
Binghamton University

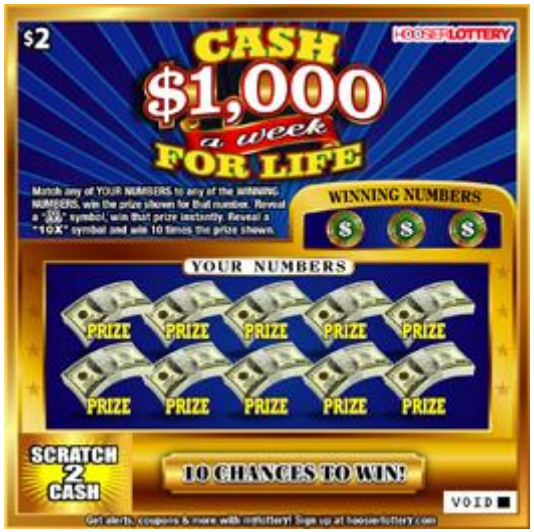
String Formatting

```
>>> price = 32.5
>>> print("${0:0.2f}".format(price))
$32.50
```

- {<index>:<width>.<precision><type>}
- <width> tells us how many spaces to use to display the value. 0 means to use as much space as necessary
- <precision> is the number of decimal places
- f means "fixed point" number

String Formatting

```
>>> "Hi {0} {1}, you won ${2}".format("Mr.",
  "Smith", 1000)
'Hi Mr. Smith, you won $1000'
```




```
>>> 'This int, {0:5}, was placed in a field of
  width 5'.format(7)
'This int,     7, was placed in a field of
  width 5'
```

String Formatting

```
>>> 'This float, {0:10.5}, has width 10 and precision 5'.format(3.1415926)
'This float,    3.1416, has width 10 and precision 5'
>>> 'This float, {0:10.5f},  is fixed at 5 decimal places'.format(3.1415926)
'This float,   3.14159, is fixed at 5 decimal places'
```

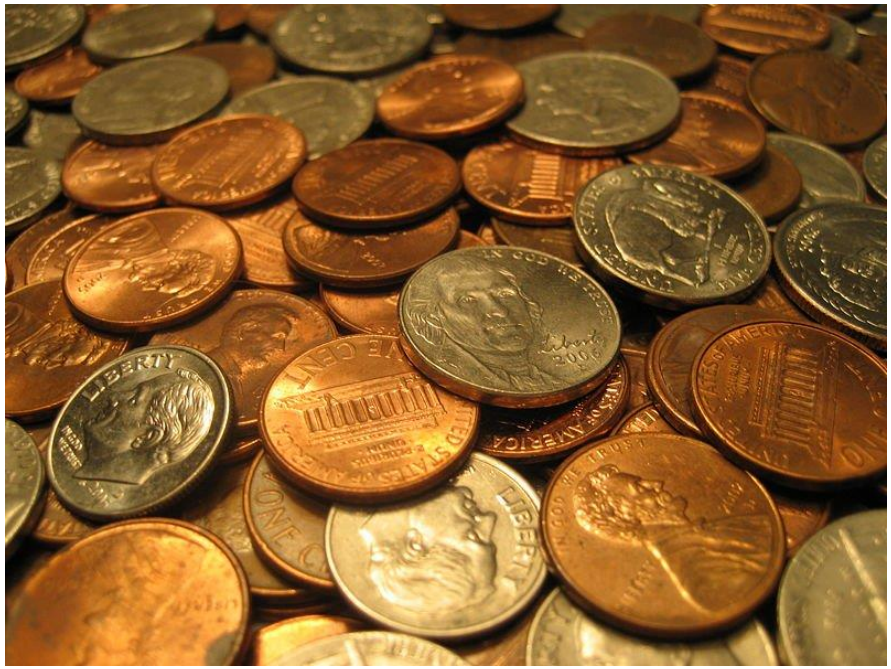
String Formatting

- If the width is wider than needed, numeric values are right-justified and strings are left-justified, by default
- You can also specify the justification before the width

```
>>>"left justification:{0:<5}".format("Hi!")
'left justification:Hi!  '
>>>"right justification: {0:0>5}".format("Hi!")
'right justification:00Hi!'
>>> "centered: {0:^5}".format("Hi!")
'centered: Hi! '
```

Change Counter

- Implement a program to calculate the value of some change in dollars



Change Counter

```
# change2.py
# A prog to calculate the value of some change in dollars

def main():
    print("Change Counter\n")
    print("Please enter the count of each type of coins.")
    quarters = eval(input("Quarters: "))
    dimes = eval(input("Dimes: "))
    nickels = eval(input("Nickels: "))
    pennies = eval(input("Pennies: "))
    total = quarters * 25 + dimes * 10 + nickels * 5 + pennies

    print("Your change is ${0:0.2f}".format(total/100))

main()
```

Change Counter

- Run this program:

```
Enter the count of each type of coins:
Quarters: 12
Dimes: 1
Nickels: 0
Pennies: 4
Your change is $3.14
```

Files: Multi-line Strings

- A *file* is a sequence of data that is stored in secondary memory (disk drive)
- Files can contain any data type, but the easiest to work with are text
- A file usually contains more than one line of text
- Python uses the newline character (\n) to mark line breaks

Multi-Line Strings

```
Hello
World

Goodbye 32
```

- When stored in a file:

```
Hello\nWorld\n\nGoodbye 32\n
```

Multi-Line Strings

```
>>> print("Hello\nWorld\n\nGoodbye 32\n")
Hello
World

Goodbye 32
```

File Processing

- Reading a file
  - File opened
  - Contents read into RAM
  - File closed

File Processing

- Working with text files in Python
  - Associate a disk file with a file object using the open function

```
<filevar> = open(<name>, <mode>)
```

  - Name is a string with the actual file name
  - The mode is either ‘r’ or ‘w’ (reading or writing)

```
file1 = open("numbers.dat", "r")
```

File Processing

- `<filevar>.read()` – returns the entire remaining contents of the file as a single (possibly large, multi-line) string
- To close an opened file, use `<filevar>.close()`

File Processing

```
# printfile.py
# Prints a file to the screen.
def main():
    fname = input("Enter filename: ")
    file1 = open(fname,'r')
    data = file1.read()
    print(data)
    file1.close()
main()
```

File Processing

- Run this program:

```
Enter filename: solar.txt
6AM      0
7AM      0.1
8AM      7.1
9AM      15.3
10AM     53.4
11AM     78.0
12PM     93.8
1PM      103.5
...(omitted)
```

File Processing

- `<filevar>.readline()` returns the next line of the file

```
>>> file1 = open("solar.txt", "r")
>>> for i in range(5):
...     line = file1.readline()
...     print(line[:-1])  #Why -1?
6AM     0
7AM     0.1
8AM     7.1
9AM     15.3
10AM    53.4
```

File Processing

- `<filevar>.readlines()` returns a list of the remaining lines

```
>>> file1 = open("solar.txt", "r")
>>> for line in file1.readlines():
...     print(line[:-1])
6AM      0
7AM      0.1
8AM      7.1
...(omitted)
```

File Processing

- Changing a file
  - Open the file in the writing mode
  - Make changes to the file
  - Close the file

File Processing

- If you open an existing file for writing, you will overwrite the file's contents
- If the named file does not exist, a new one is created

```
>>> outfile = open("myout.txt", "w")
>>> print("Hello\nWorld\n\nGoodbye", file=outfile)
>>> outfile.close()
```

Example: Batch Usernames

- *Batch* mode processing is where program input and output are done through files (i.e., the program is not designed to be interactive)
- Example: Create usernames for a computer system where the first and last names come from a file

Example: Batch Usernames

```
# userfile.py
def main():
    print("Create a file of usernames in batch mode.")
    infileName =input("What file are the names in?")
    infile = open(infileName, 'r')

    outfileName=input("What file should the usernames go in?")
    outfile = open(outfileName, 'w')

    # process each line of the input file
    for line in infile:
        first, last = line.split()
        uname = (first[0]+last[:7]).lower()
        print(uname, file=outfile)
    infile.close()
    outfile.close()
    print("Usernames have been written to", outfileName)

main()
```

File Processing

- Run the program:

```
Create a file of usernames in batch mode.
What file are the names in?peoplenames.txt
What file should the usernames go in?myout.txt
Usernames have been written to myout.txt
```

peoplenames.txt - C:\...

File Edit Format Run Options Window Help

```
Hillary Clinton
Bernie Sanders
Jeb Bush
Ted Cruz
Donald Trump
Michael Bloomberg
```

myout.txt - C:\Users\ted...

File Edit Format Run Options Window Help

```
hclinton
bsanders
jbush
tcruz
dtrump
mbloombe
```