Formatted Output

CSE 1310 – Introduction to Computers and Programming

Importance of Formatting

- When we print information on the screen, formatting can make a big difference:
- Nicely formatted printouts are much easier for a user to read.
- For example: in printing a matrix:

```
[[3.394012, 0.21312, 0.90213232], [9.12, 12.30412,
1], [32.34, 1.232, 0.2]]
```

Importance of Formatting

- When we print information on the screen, formatting can make a big difference:
- Nicely formatted printouts are much easier for a user to read.
- For example: in printing a matrix:

```
 3.394    0.213    0.902
 9.120   12.304    1.000
32.340    1.232    0.200
```

The **format** Method for Strings

- my_string.format(item1, item2, …, item_n)
- Looks for placeholders, marked by {} in my_string, and replaces them with the specified items.
- Simple example:

```
my_string = "{} is today, and {} is tomorrow"
print(my_string.format("Tuesday", "Wednesday"))

OUTPUT:
'Tuesday is today, and Wednesday is tomorrow'
```

```
matrix1 = [[3.394012, .21312, .90213232],
           [9.12, 12.30412, 1],
           [32.34, 1.232, 0.2]]

print(matrix1)
```

```
OUTPUT:
[[3.394012, 0.21312, 0.90213232], [9.12, 12.30412,
1], [32.34, 1.232, 0.2]]
```

- Example 1: no formatting here, just printing the matrix as a list.

```
matrix1 = [[3.394012, .21312, .90213232],
           [9.12, 12.30412, 1],
           [32.34, 1.232, 0.2]]

for item in matrix1:
    print(item)

OUTPUT:
[3.394012, 0.21312, 0.90213232]
[9.12, 12.30412, 1]
[32.34, 1.232, 0.2]
```

- Example 2: again no formatting, just printing each line separately.

```
matrix1 = [[3.394012, .21312, .90213232],
           [9.12, 12.30412, 1],
           [32.34, 1.232, 0.2]]

for item in matrix1:
    print("{:> 11} {:> 11} {:> 11}".format(item[0],
item[1], item[2]))

OUTPUT:
   3.394012     0.21312  0.90213232
       9.12    12.30412           1
      32.34       1.232         0.2
```

- Example 3: Here we use formatting:
  - :> means "right alignment"
  - 11 means "minimum width of 11 for that item"

```
matrix1 = [[3.394012, .21312, .90213232],
           [9.12, 12.30412, 1],
           [32.34, 1.232, 0.2]]

for item in matrix1:
    print("{:> 5} {:> 5} {:> 5}".format(item[0], item[1], 
item[2]))

OUTPUT:
 3.394012  0.21312  0.90213232
 9.12  12.30412     1
 32.34  1.232   0.2
```

- Example 4: Here we change the previous example, by setting the minimum width to 5 instead of 11:
  - The columns are not aligned anymore.
  - Why?

```
matrix1 = [[3.394012, .21312, .90213232],
           [9.12, 12.30412, 1],
           [32.34, 1.232, 0.2]]

for item in matrix1:
    print("{:> 5} {:> 5} {:> 5}".format(item[0], item[1],
item[2]))

OUTPUT:
 3.394012  0.21312  0.90213232
 9.12  12.30412     1
 32.34  1.232   0.2
```

- Example 4: Here we change the previous example, by setting the minimum width to 5 instead of 11:
  - The columns are not aligned anymore.
  - Why? Because some items need more than the minimum width.

```
matrix1 = [[3.394012, .21312, .90213232],
           [9.12, 12.30412, 1],
           [32.34, 1.232, 0.2]]

for item in matrix1:
    print("{:> 5} {:> 5} {:> 5}".format(item[0], item[1],
item[2]))

OUTPUT:
 3.394012  0.21312  0.90213232
 9.12  12.30412     1
 32.34  1.232   0.2
```

- Example 4: What is a good value for the minimum width?

```
matrix1 = [[3.394012, .21312, .90213232],
           [9.12, 12.30412, 1],
           [32.34, 1.232, 0.2]]

for item in matrix1:
    print("{:> 5} {:> 5} {:> 5}".format(item[0], item[1],
item[2]))

OUTPUT:
 3.394012  0.21312  0.90213232
 9.12  12.30412     1
 32.34  1.232   0.2
```

- Example 4: What is a good value for the minimum width? It should be large enough to accommodate the longest item.

```
matrix1 = [[3.394012, .21312, .90213232],
           [9.12, 12.30412, 1],
           [32.34, 1.232, 0.2]]

for item in matrix1:
    print("{:> 11f} {:> 11f} {:> 11f}".format(item[0],
item[1], item[2]))

OUTPUT:
   3.394012    0.213120    0.902132
   9.120000   12.304120    1.000000
  32.340000    1.232000    0.200000
```

- Example 5: we use {:> 11f}.
- f specifies that the item should be printed as a float.
- In floats, by default six decimal places are printed.

```
matrix1 = [[3.394012, .21312, .90213232],
           [9.12, 12.30412, 1],
           [32.34, 1.232, 0.2]]

for item in matrix1:
    print("{:> 8.3f} {:> 8.3f} {:> 8.3f}".format(item[0],
item[1], item[2]))

OUTPUT:
   3.394    0.213    0.902
   9.120   12.304    1.000
  32.340    1.232    0.200
```

- Example 6: {:> 8.3f}.
- 8.3f means: print a float, with a minimum width of 8, and 3 decimal places printed for each float.

Specifying **format**

- While there can be more complicated versions (see textbook Section 4.7), the most common **format** specifications are these:

{:[align] [minimum_width] [.precision] [descriptor]}

Square brackets indicate optional parts (that may be skipped).

Specifying **format**

- While there can be more complicated versions (see textbook Section 4.7), the most common **format** specifications are these:

{:[**align**] [minimum_width] [.precision] [descriptor]}

- The alignment can be of three types:
  - < for left alignment
  - > for right alignment
  - ^ for center alignment

Specifying **format**

- While there can be more complicated versions (see textbook Section 4.7), the most common **format** specifications are these:

{:[align] [minimum_width] [.precision] [**descriptor**]}

- The most common descriptors are:
  - s for string
  - d for integer
  - f for floating-point number
  - e for floating point number in scientific (exponential) notation
  - % for floating point number as percent.

Example: Months and Days

```
month_names = ["January", "February", "March",
               "April", "May", "June", "July",
               "August", "September", "October",
               "November", "December"]

month_lengths = [31, 28, 31, 30, 31, 30,
                 31, 31, 30, 31, 30, 31]

for i in range(0, 12):
    print("{:>13s}: {:>2} days".format(month_names[i],
month_lengths[i]))
```

Example: Months and Days: Output

```
  January: 31 days
 February: 28 days
    March: 31 days
    April: 30 days
      May: 31 days
     June: 30 days
     July: 31 days
   August: 31 days
September: 30 days
  October: 31 days
 November: 30 days
 December: 31 days
```