Lecture 11 - Formatted printing

Computer Programming

Robert D. Vincent and Samia Hilal

Marianopolis College

April 6, 2022

Why use formatting?

- Printing numbers of varying length:

```
matrix = [[10,2,3],[2,300,1],
          [-100,5,2]]
for row in matrix:
    for val in row:
        print(val, end=' ')
    print()
```

- This prints:

```
10 2 3
2 300 1
-100 5 2
```

- Resulting text is hard to interpret.

Why use formatting?

- Most languages let us format text nicely:

```
matrix = [[10,2,3],[2,300,1],
          [-100,5,2]]
for row in matrix:
    for val in row:
        print('{:4d}'.format(val),
              end=' ')
    print()
```

- This looks a bit better:

```
  10     2     3
   2   300     1
-100     5     2
```

Approaches to formatting

- Python offers two different approaches to formatting text.
  1. The format operator '%'
  2. The `format()` method.
- The format operator is associated with Python 2.X, but still supported in 3.X
- Both use a format string and a list of arguments to produce an output string.
- The `format()` method is newer and more flexible, so we will concentrate on it.

Format string

- A pair of curly braces defines a *replacement field*.
- Each replacement field is replaced with an argument.
- Empty replacement fields use the default formatting for the argument.
- We can supply *format specifications* that replace or augment the defaults.
- Characters outside the curly braces are copied to the output string.
- Curly braces may be included in the result by doubling them: `'{{'` or `'}}'`.

Simple example

```
x = 9
y = 3
fmt_str = '{} ** {} is {}.'
msg_str = fmt_str.format(x, y, x ** y)
print(msg_str)
```

- This program will print:

```
9 ** 3 is 729.
```

- Format creates a new string.
- The first **{}** is replaced by the argument `x`, etc.

Replacement fields

- Two parts, separated by a colon `':'`.
- The first part is an argument index that selects *which* argument to format.
- The second part, the *format specification* tells `format()` how to convert the selected argument.

```
>>> L = [4.1, 3.8, 2.9]
>>> x = sum(L)/len(L)
>>> print('The mean is {0:f}'.format(x))
The mean is 3.600000
```

The first part of the replacement field

- One common case: a numeric argument index in the curly braces:

```
>>> x, y = 'parrot', 'bird'
>>> print('A {} is a {}'.format(x, y))
A parrot is a bird
>>> print('A {0} is a {1}'.format(x, y))
A parrot is a bird
>>> print('A {1} is a {0}'.format(x, y))
A bird is a parrot
```

- The number, if present, tells `format()` which argument to use.

The first part, continued

- An element index corresponds to the order of the arguments to `format()`. They can be re-used:

```
>>> print('{0}{1}{0}'.format('abra', 'cad'))
abracadabra
```

- The first part can also specify named elements or indices in a sequence.

Element index examples

```
>>> print('{}{}{}'.format('a', 'b', 'c'))
abc
>>> print('{0}{1}{2}'.format('a', 'b', 'c'))
abc
>>> print('{2}{0}{1}'.format('a', 'b', 'c'))
cab
>>> print('Enter {1} numbers:'.format(3, 4))
Enter 4 numbers:
```

The format specification

If present, appears after the optional ':' character, including:

1. Fill character.
2. Alignment option.
3. Sign-handling option.
4. **Width.**
5. **Precision.**
6. **Conversion type.**

All of the fields are optional, but the order is important. We will only really cover the ones in **bold**.

Format specification

- Fill character - allows you to use a character other than space to 'pad' the width.
- Alignment option - left-justify, right-justify, or center a field.
- Sign-handling option - how to indicate sign of numbers.

Conversion types

Character	Meaning	Type
`d`	Decimal integer	int
`b`	Binary integer	int
`x` or `X`	Hexadecimal integer	int
`o`	Octal integer	int
`e` or `E`	Floating point scientific	Number
`f` or `F`	Floating point decimal	Number
`g` or `G`	Floating point general	Number
`%`	Percentage	Number
`c`	A single character	int
`s`	A string	str

- Capital letters use upper case in numbers.

Width

- Specifies the *minimum* width for the field:

```
w, x, y = 125, 125.9, 'Hello'
>>> print('/{:2d}/'.format(w))
/125/
>>> print('/{:5d}/'.format(w))
/  125/
>>> print('/{:10s}/'.format(y))
/Hello     /
>>> print('/{:12f}/'.format(x))
/  125.900000/
```

- Strings are left-justified by default.
- Numbers are right-justified by default.

Precision

- Separated from width by a period (`'.'`)
- For 's', the *maximum* characters to print.
- For 'e' and 'f', number of fractional digits.
- For 'g', maximum total digits.
- Not allowed with integer conversions (e.g. 'd').
- Can be combined with width.

Precision with strings

```
x = 'Goodbye'
>>> print('/{:.2s}/'.format(x))
/Go/
>>> print('/{:.4s}/'.format(x))
/Good/
>>> print('/{:6.4s}/'.format(x))
/Good  /
>>> print('/{:10.10s}/'.format(x))
/Goodbye   /
```

Precision with floats

```
>>> x, y, z = -2.34567, 2.34567, 1.5e6
>>> print('/{:5.2f}/{:5.2f}/'.format(x, y))
/-2.35/ 2.35/
>>> print('/{:5.2e}/{:5.2e}/'.format(x, y))
/-2.35e+00/2.35e+00/
>>> print('/{:5.2g}/{:5.2g}/'.format(x, y))
/ -2.3/  2.3/
>>> print('{:6.4f} {:6.4E}'.format(y, z))
2.3457 1.5000E+06
>>> print('{:.3f} {:.3g}'.format(z, z))
1500000.000 1.5e+06
```

Summary

- The `format()` method generates a string using a format string and its other arguments.
- Provides very detailed control over how your data is converted and printed.
- I don't expect you to memorize all of the details!
- You should understand how to select specific arguments to format using an argument index.
- You should be familiar with the basic conversion types ('d', 'f', 'e', 's'), the width, and the precision.

Alignment option examples

```
>>> print('/{:5}/'.format(-10))
/  -10/
>>> print('/{:<5}/'.format(-10))
/-10  /
>>> print('/{:^5}/'.format(-10))
/ -10 /
```

Sign handling examples

```
>>> print('/{:5d}/'.format(10))
/   10/
>>> print('/{:5d}/'.format(-10))
/  -10/
>>> print('/{:+5d}/'.format(10))
/  +10/
>>> print('/{:+5d}/'.format(-10))
/  -10/
>>> print('/{:=5d}/'.format(-10))
/-   10/
```

Other examples

```
>>> print('{:b}'.format(120))
1111000
>>> print('{:o}'.format(120))
170
>>> print('{:04x}'.format(120))
0078
>>> print('{:#06x}'.format(120))
0x0078
>>> print('{:c}'.format(120)) # ascii 120
x
>>> print('{0:,}'.format(1000000))
1,000,000
>>> print('{:.2%}'.format(.99))
99.00%
```