CSC 223 - Advanced Scientific Programming

Python Input and Output

Formatting Output

- Python string objects have methods for formatting output:
  - `center`
  - `format`
  - `ljust`
  - `rjust`
  - `zfill`

Formatting Example

```
>>> for x in range(1,11):
...     print(str(x).rjust(2), end=' ')
...     print(str(x*x).rjust(3), end=' ')
...     print(str(x*x*x).rjust(4))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

`format` Basics

- The `format` method requires that the string have `format fields` indicated with curly braces.
- The format fields are replaced with objects passed into the `format` method.
- Example:

```
>>> '{} and {}'.format('Alice', 'Bob')
'Alice and Bob'
```

format Arguments

- The format fields can be replaced by position or keyword
- Position example:

```
>>> '{1} and {0}'.format('Alice', 'Bob')
'Bob and Alice'
```

- Keyword example:

```
>>> '{a} and {b}'.format(a='Alice', b='Bob')
'Alice and Bob'
```

- Mixed example:

```
>>> '{b} and {0}'.format('Alice', b='Bob')
'Bob and Alice'
```

Formatting Example with `format`

```
>>> for x in range(1,11):
...     print('{:2} {:3} {:4}'.format(x,x**2,x**3))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

`format` Specifiers

- An optional `':'` and format specifier can follow the format field name
- Format specifiers enable greater control over formatted values
- Syntax (all are optional): {: [1] [2] [3] [4] [5] [6] [7] [8] }
  1. fill and alignment
  2. sign
  3. `'#'` – alternate number form
  4. `'0'` – sign aware zero padding
  5. width
  6. grouping
  7. '.' followed by integer – precision
  8. type

format Specifiers: Width Option

- The width option is a positive integer that specifies padding
- If the width is too small, nothing happens
- Example:

```
>>> '{:10}'.format('hello')
'hello     '
>>> '{:2}'.format('hello')
'hello'
```

`format` Specifiers: Fill and Alignment

- Alignment options:
  - `'<'`: left align
  - `'>'`: right align
  - `'='`: pad after sign (if any) but before digits
  - `'^'`: center align
- A fill character can optionally be specified before the alignment character
- Example:

```
>>> '{:^11}'.format('hello')
'   hello   '
>>> '{:>11}'.format('hello')
'      hello'
>>> '{:*>11}'.format('hello')
'******hello'
>>> '{:-^11}'.format('hello')
'---hello---'
```

`format` Specifiers: Sign option

- The sign option is only valid for number types
  - `'+'`: sign should be used for both positive and negative numbers
  - `'-'`: sign should be used for only negative numbers (default)
  - `' '` (space): leading space for positive numbers and minus sign for negative numbers
- Examples

```
>>> '{:+}'.format(123)
'+123'
>>> '{: }'.format(123)
' 123'
>>> '{:+}'.format(1.414)
'+1.414'
```

format Specifiers:

- The # option causes a type specific “alternate form” to be used for the conversion
- The # can only be used for integer, float, complex, and Decimal types
- Example

```
>>> # print 123 in binary
>>> '{:b}'.format(123)
'1111011'
>>> '{:#b}'.format(123)
'0b1111011'
```

`format` Specifiers: 0

- The 0 (preceding the width option) enables sign aware zero-padding for numeric types
- Example

```
>>> '{:010}'.format(123)
'0000000123'
>>> '{:010}'.format(1.414)
'000001.414'
```

`format` Specifiers: Grouping Option

- The grouping option specifies a character for separating thousands in numbers
- The grouping option can be either '_' or ','
- Example

```
>>> '{:_}'.format(1000000)
'1_000_000'
>>> '{:,}'.format(1000000)
'1,000,000'
```

`format` Specifiers: Precision Option

- The precision option specifies how many digits to be displayed:
  - after the decimal point for fixed point floating point values
  - before and after the decimal point for general floating point values
- Example

```
>>> x = math.sqrt(2)
>>> '{:.2}'.format(x)
'1.4'
>>> '{:.2f}'.format(x)
'1.41'
>>> '{:.6}'.format(x)
'1.41421'
>>> '{:.6f}'.format(x)
'1.414214'
```

`format` Specifiers: Type Option

- String option:
  - `'s'`: string
- Common integer options:
  - `'b'`: binary
  - `'c'`: character
  - `'d'`: decimal
  - `'o'`: octal
  - `'x'`: hex (lower case)
  - `'X'`: hex (upper case)
- Common float options:
  - `'e'`: exponent notation
  - `'E'`: exponent notation (upper case E)
  - `'f'`: fixed point
  - `'F'`: fixed point (upper case NAN and INF)
  - `'g'`: general format
  - `'%'`: percent (multiply by 100)

format Specifiers: Type Option

- Integer examples

```
>>> '{:d}'.format(123)
'123'
>>> '{:b}'.format(123)
'1111011'
>>> '{:X}'.format(123)
'7B'
```

- Floating point examples

```
>>> '{:f}'.format(1.414)
'1.414000'
>>> '{:E}'.format(1.414)
'1.414000E+00'
>>> '{:%}'.format(1.414)
'141.400000%'
```

File IO

- The `open` function returns a file object.
- Basic syntax
  `open(`*`filename`*`, `*`mode`*`)`
- The *mode* parameter is a string that specifies permissions:
  - `'r'` – read
  - `'w'` – write (existing file with same name is erased)
  - `'a'` – append
  - `'r+'` – read and write

Basic Methods on File Objects

- `read` – read the contents of a file
- `readline` – read a single line from the file
- `write` – write to a file
- `close` – close the file
- File objects can be treated as iterators

File Example

```
# copy the contents of one file to another
inFile = open('myfile_in')
outFile = open('myfile_out', 'w')

for line in file:
    outFile.write(line)

inFile.close()
outFile.close()
```

The `with` Keyword

- The `with` keyword can be used for a category of objects called *context managers*.
- A context manager has implementations of the `__enter__` and `__exit__` methods.
- The `with` statement guarantees that the `__exit__` method is called at the end of the block.
- The `__exit__` method of a file object closes the file.
- Example:

```
with open('myfile.txt', 'w') as f:
    f.write('Hello world!')
# the file is guaranteed to be closed here
```

File Formats

- Python has many modules to handle many file formats commonly used in data science tasks:
  - Comma separated values (CSV)
  - Hypertext Markup Language (HTML)
  - Extensible Markup Language (XML)
  - JavaScript Object Notation (JSON)
  - Sqlite database files