12.010 Computational Methods of Scientific Programming

Lecture 13: Outputting and formatting results.

Summary

- Output results: formatting, table structures, outputting for papers and reports.
- Formatting in Python output
    - Styles of syntax: f and % versions
    - Number formats
    - Special characters (tabs, vt100 escape sequences)
- Panda data frame formatting options
- Formatting in Notebooks
- Writing output to files (for use in other applications).

str.format methods

- Under the str class, there is a very flexible str.format method that allows control over how results are output.
- The grammar for a replacement field is as follows:
- **replacement_field** ::= "{" [field_name] ["!" conversion] [":" format_spec] "}"
  field_name ::= arg_name ("." attribute_name | "[" element_index "]")*
  arg_name ::= [identifier | digit+]
  attribute_name ::= identifier
  element_index ::= digit+ | index_string
  index_string ::= <any source character except "]"> + **conversion** ::= "r" | "s" | "a"
  format_spec ::= <described in the next section>

Replacement field*

- Contained in {} and can be arguments (starting at zero) from a sequence or object names
- The conversion entry is optional but can be:
  - !s – str() method
  - !r – repr() method: Return a string containing a printable representation of an object; may be passed to eval(). (__repr__ method can be included in class to control how this is done (more on classes later).
  - !a – ascii() method: return a string containing a printable representation of an object, but escape the non-ASCII characters in the string using \x, \u or \U

Format Specification

- The general form of a *standard format specifier* is:
- **format_spec** ::=
  [[fill]align][sign][#][0][width][grouping_option][.precision][type]

  fill ::= <any character>
  align ::= "<" | ">" | "=" | "^" – alignment ^ is centered in field
  sign ::= "+" | "-" | " " – sets how +- are displayed
  width ::= digit+
  grouping_option ::= "_" | ","   -- sets thousand separator.
  precision ::= digit+
  type ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"

Type options*

- Integer types

Type	Meaning
'b'	Binary format. Outputs the number in base 2.
'c'	Character. Converts the integer to the corresponding unicode character before printing.
'd'	Decimal Integer. Outputs the number in base 10.
'o'	Octal format. Outputs the number in base 8.
'x'	Hex format. Outputs the number in base 16, using lower-case letters for the digits above 9.
'X'	Hex format. Outputs the number in base 16, using upper-case letters for the digits above 9. In case '#' is specified, the prefix '0x' will be upper-cased to '0X' as well.
'n'	Number. This is the same as 'd', except that it uses the current locale setting to insert the appropriate number separator characters.
None	The same as 'd'.

Older C-like methods

- Use of print with %<format> options with % and tupple with arguments to be printed.
- This is an older method but still usable (not deprecated yet).
- Example in the notebook and we have used this in other codes.

Control characters*

- Control characters can be added to print formats as well
- Main codes:
  - \n new-line (not needed unless end=‘’ used
  - \r return (no newline – see notebook for one way to use.
  - \t tab (useful if writing tables to be converted to a table in Word)
  - \\ To output \

IO to file*

- File can be opened with open
  wf = open('word.txt','w’)
- Then:
  - wf.read(size) – reads size characters from the file. Reads whole file if size is negative or not given. File read into string
  - wf.readline – reads next line in file
  - wf.write(str) – writes line to file. (Same constructs as print).
  - wf.close() – closes file and writes remaining part of file to ‘disk’

tabulate

- Needs:
  conda install tabulate

```
 package                    |            build
 ---------------------------|-----------------
 conda-4.10.3               |   py38hecd8cb5_0         2.9 MB
 tabulate-0.8.9             |   py38hecd8cb5_0          40 KB
 ------------------------------------------------------------
                                        Total:          2.9 MB
```

Markdown formatting.

- Material from https://www.markdownguide.org/basic-syntax/
- Headings (space after #..# needed)
- Alternative for h1 and h2 add ===== or ----- below line

Markdown	HTML	Rendered Output
# Heading level 1	<h1>Heading level 1</h1>	Heading level 1
## Heading level 2	<h2>Heading level 2</h2>	Heading level 2
### Heading level 3	<h3>Heading level 3</h3>	Heading level 3
#### Heading level 4	<h4>Heading level 4</h4>	Heading level 4
##### Heading level 5	<h5>Heading level 5</h5>	Heading level 5
###### Heading level 6	<h6>Heading level 6</h6>	Heading level 6

Paragraphs/Lines/Emphasis

- Add blank line between blocks of text.
- To get line break (lines by default are concatenated together to span the width of the notebook), add 2 or more spaces at the ends of line or use <br>.
- Bold uses **text** or __text__
- Italics uses *text* or _text_
- Bold Italics uses ***text*** or ___text___
- When no spaces _ acts differently to *

Blockquotes/Lists

- Use > in start of lines for block quote
- Ends with new paragraph unless the blank line between paragraphs starts with > as well.
- Use >> to indent nested blockquotes.
- Lists: Just need to start with numeric value (with decimal point), list will increment no matter what values are used.
- If decimal point is needed one a first entry, ‘escape’ it with \ i.e., \.
- Indent 4-spaces in list with create paragraph indented but with no leading bullet.

Code blocks

- Sometimes you don't want fancy formatting and just want to show simple text or code. Indenting by 4-spaces of \<tab> will do this.
- (HW01 solution used a raw block to this but the methods below allow other formatted text to be added.
- 4-spaces or tab at start of new paragraph created code block.
- Enhanced method is to used ~~~ to start and end blocks.
- Syntax can be recognized by added language name after ~~~ e.g., ~~~python.

Imbed images and URLs

- Syntax for images is:
  ![label] (image file name)
- Many implementations don’t allow image size change, in which case html code can be used. (See notebook).

Tables

- Basic syntax is to use --- and | to show where rules should go.
- :--, --: and :--: set left, right and center justifications.
- Table can also be created with html syntax.

Latex in Markdown

- Latex equation syntax can be used in Markdown
- In notebook: editing seems to need cell to be changed to code and then Markdown so that equations will be rendered after editing.
- These equations can often be imported directly into Word Equations as well e.g.

$$\int_{\Omega} \nabla u \cdot \nabla v \sim dx \; = \; \int_{\Omega} f v \sim dx$$

Summary

- Formatting is very flexible but can be confusing with the different options for doing the same thing. Methods also change with releases.
- Learning number and table formatting can be useful when preparing tables for inclusion in Word and LaTeX documents.
- Markdown language can be used when Notebooks are distributed to other