The Python Standard Library: part 2

Prof. Pai H. Chou
National Tsing Hua University

Python Standard Library cont'd

- Text Processing
- File access, data persistence, compression, format, cryptography
- OS service, concurrent execution
- Networking, HTML, XML
- Multimedia, internationalization
- Framework for graphic, GUI, command-line

Text processing modules

- string — Common string operations
- re — Regular expression operations
- difflib — Helpers for computing deltas
- textwrap — Text wrapping and filling
- unicodedata — Unicode Database
- stringprep — Internet String Preparation
- readline — GNU readline interface
- rlcompleter — Completion function for GNU readline

string module: constants

- Constants
  - ascii_letters
  - ascii_lowercase
  - ascii_uppercase
  - digits
  - hexdigits
  - octdigits
  - punctuation
  - printable
  - whitespace
  - Usage:
    - check char in const

```
>>> import string
>>> 'a' in string.ascii_lowercase
True
>>> string.hexdigits
'0123456789abcdefABCDEF'
```

string module: Template class

- Template class
  - substitutes $variables with their values

```
>>> from string import Template
>>> s = Template('$who likes $what')
>>> s.substitute(who='tim', what='kung pao')
'tim likes kung pao'
```

  - alternative string format, similar to shell or Perl

Regular Expressions

- a pattern-matching "mini-language"
- Line editors
  - ex, vi, vim, some IDE text editors, though somewhat different
- Batch processing programs (Unix shell commands)
  - sed: search/replace lines of text that match pattern
  - awk: C-like scripting language to control regular expressions
  - egrep: displays lines that match the specified pattern
  - perl: super awk
  - Python: `re` module

Finding a word in vi/vim: /pattern

- Literal match:
  - e.g., `/or` matches or, world, factor, ...
- Pattern match
  - `/\<or\>` \< \> (in vim) matches whole word ("or")
  - `/\<or` \< (in vim) matches word beginning ("orange")
  - `/or\>` \> (in vim) matches word ending ("factor")
  - `/^or` ^ matches beginning of line
  - `/or$` $ matches end of line

vim search-replace example

```
  1 1-2-2018
  2 2-3-2018
  3 12-21-2019
  4 10-5-2019
  5 07-11-1983
  6
~
~
~
~
:%s/\([0-9][0-9]*\)-\([0-9][0-9]*\)-\([0-9][0-9]*\)/\2-\1-\3/g
```

```
  1 2-1-2018
  2 3-2-2018
  3 21-12-2019
  4 5-10-2019
  5 11-07-1983
  6
~
~
~
5 changes; after #2
```

vim search/replace explained

```
:%s/\([0-9][0-9]*\)-\([0-9][0-9]*\)-\([0-9][0-9]*\)/\2-\1-\3/g
```

`:%` "on all lines"

"globally" = as many times as can match

`:%s/` *pattern to search* `/` *replace* `/g`

`:%s/\(` *1st pattern* `\)` `\(` *2nd pattern* `\)` `\(` *3rd pattern* `\)/\2` `\1` `\3/g`

`[0-9]` matches one character in {'0', '1', '2', ... '9'} (decimal digit)

`[0-9]*` matches a string with zero or more digits

`[0-9][0-9]*` matches a string with one or more digits

Alternatively, `\d` is another way to say `[0-9]`

```
:%s/-/-/\2-\1-\3/g
```

`-` matches -

`-` replaces with -

Alternative vim search/replace

```
:%s/\([0-9][0-9]*\)-\([0-9][0-9]*\)-\([0-9][0-9]*\)/\2-\1-\3/g
```

```
:%s/\([01]\d\)-\([0-3]\d\)-\(\d{4}\)/\2-\1-\3/g
```

- issue: may want to limit # of digits per field
  - exactly 4 digits for year
  - 1-2 digits for month and year
  - [01] for 10-month, [0-3] for 10-day
- `[01]\d` 00,01,..10,11,..19

  `[0-3]\d` 00,01,..10,11,..39

  `\d{4}` four digits

Python re module

- "pattern matching"
  - `search()`, `findall()`: anywhere in string
  - `match()`, `fullmatch()`: from beginning or whole string
- `split()`: string by separator pattern
- `sub()`:"substitute" = search and replace

Python re module

- `search()`: return Match object for first found
- `findall()`: return list of found strings

```
>>> import re
>>> s = re.search('Gary', 'Mary had a little lamb')
>>> s   # if not found, s is None
>>>
>>> s = re.search('little', 'Mary had a little lamb')
>>> s
<re.Match object; span=(11, 17), match='little'>
>>> s.group(0)
'little'
>>> s.span(0)
(11, 17)
>>> re.findall('little', 'Mary had a little lamb, little lamb')
['little', 'little']
```

re character pattern

pattern	meaning
.	any character
[abc]	set of characters
[^abc]	NOT abc
[a-z]	character range

```
>>> # want to match both Mary and Gary? use .
>>> re.findall('.ary', 'Mary and Gary had a little lamb')
['Mary', 'Gary']
>>> # but . may be too general and match words we don't want
>>> re.findall('.ary', 'Mary and Gary think a lamb is scary')
['Mary', 'Gary', 'cary']
>>> # solution: limit the set of characters to match
>>> re.findall('[A-Z]ary', 'Mary and Gary think a lamb is scary')
['Mary', 'Gary']
```

re pattern repetition

pattern	meaning
*	0 or more occurrences
?	0 or 1 occurrence
+	1 or more occurrences

pattern	meaning
{*n*}	*n* occurrences
{*m*,*n*}	*m*..*n* occurrences (inclusive)

```
>>> # want to match longer words, put * after the char set
>>> re.findall('[a-z]*are', 'We scare because we care')
['scare', 'care']
>>> re.findall('[a-z]*are', 'We are here to scare and care')
['are', 'scare', 'care']
>>> # to exclude "are", use + instead of *
>>> re.findall('[a-z]+are', 'We are here to scare and care')
['scare', 'care']
>>> re.findall('[a-z]+are', 'He cares that we are scared')
['care', 'scare']     # without s and d. to include, use ?
>>> re.findall('[a-z]+are[sd]?', 'He cares that we are scared')
['cares', 'scared']
```

blanks and word boundaries

- \ is re notation
  - but \ is special in regular python string
  - Python string literal `'\\s'` to mean `\s` str value (because \ needs to be escaped by another \ )

pattern	meaning	equivalent set
`\s`	blanks	`[ \t\n\r\f\v]`
`\b`	word boundary	🚫
`\w`	word character	`[A-Za-z0-9_]`
`^`	beginning of line	🚫
`$`	end of line	🚫

- Solution: raw string so that \ is a literal
  - raw string `r'\s'` is the same as `'\\s'`
  - => much easier to read

blank and word boundary

```
>>> re.findall(r'\w+are[sd]?\b', 'He cares that we are scared')
['cares', 'scared']
```

- \w+ matches "word" characters (alphanumeric,_)
- \b matches word boundary (but not actual char)

```
>>> re.findall(r'\bMary\b',"Mary and Mary's lamb like Mary")
['Mary', 'Mary', 'Mary']
```

- ^ = beginning of line
- $ = end of line

```
>>> re.findall(r'^Mary\b', "Mary and Mary's lamb like Mary")
['Mary']
>>> re.search(r'\bMary$', "Mary and Mary's lamb like Mary")
<re.Match object; span=(26, 30), match='Mary'>
```

pat.	meaning
\s	blanks
\b	boundary
\w	word char
^	line beg.
$	line end

Character patterns for Python re

meaning	regular expr.	equivalent ASCII set
word boundary	`\b`	🚫
not beginning or end of word	`\B`	🚫
decimal digits	`\d`	`[0-9]`
not decimal digit	`\D`	`[^0-9]`
whitespace	`\s`	`[ \t\n\r\f\v]`
not whitespace	`\S`	`[^ \t\n\r\f\v]`
an alphanumeric character	`\w`	`[a-zA-Z0-9_]`
not alphanumeric character	`\W`	`[^a-zA-Z0-9_]`
beginning of string	`\A`	🚫
end of string	`\Z`	🚫

* and + (Greedy) vs. *? and *+

- * and + longest match possible (greedy)
  - tends to match more than you think...

```
>>> re.findall(r'\b.*are', 'He cares that we are scared')
['He cares that we are scare']
```

```
>>> re.findall(r'\b.+are', 'He cares that we are scared')
['He cares that we are scare']
```

- Solution: non-greedy match *? and +?

```
>>> re.findall(r'\b.*?are', 'He cares that we are scared')
['He care', ' that we are', ' scare']
```

```
>>> re.findall(r'\b.+?are', 'He cares that we are scared')
['He care', ' that we are', ' scare']
```

string splitting

- str split method: default splits on blanks

```
>>> "Mary had a little lamb".split() # default splits on \s+
['Mary', 'had', 'a', 'little', 'lamb']
>>> "Mary  \v\f\n  had \n  a \t\r little lamb".split()
['Mary', 'had', 'a', 'little', 'lamb']
```

  - this is effectively splitting on re `r'\s+'`

- can split on single- or multi-char separator

```
>>> 'Mary//had//a//little//lamb'.split('//')
['Mary', 'had', 'a', 'little', 'lamb']
```

  - however, str's split separator must be constant strings, not regular expressions!

```
'  Mary  //  had    // a    // little  //lamb'
```

Separators that cannot be handled by str's split() method

- combination of punctuations and blanks
  - example string: "To be, or not to be -- that is the question!"
    - => want the result list to be just words, no space or punctuation
    - separator has pattern `r'\W+'`
- Solution: use `re.split()`

re.split()

- usage: re.split(pattern, stringTarget)

```
>>> re.split(r'\W+', "To be, or not to be--that is the question!")
['To', 'be', 'or', 'not', 'to', 'be', 'that', 'is', 'the',
'question', '']
```

- Can handle combination of separators
  - e.g., using // with optional blanks
  - string: 'hello//world //what// do // you // think'

```
>>> re.split(r'\s*//\s*', 'hello//world    //how// are  //   you')
['hello', 'world', 'how', 'are', 'you']
```

more example splitting with re

- Example: split on blanks or HTML tags
  - blanks = `\s+`
  - HTML tag = `<.*?>` # note: *? is nongreedy match
  - use `|` for matching either pattern

```
>>> re.split(r'\s+|<.*?>', "<html>foo<a>bar</a>end</html>")
['', 'foo', 'bar', 'end', '']
```

  - more powerful than str's split() method

Concept of grouping

- use `(  )` in re to enclose the pattern to group
  - Capture a pattern
  - but... `(? ... )` are special. e.g., `(?# comment)`
- Ways to reference a group
  - Match object can use `.group(i)` to get group content
  - re itself can also reference `\1`, `\2`, `\3`.... as group content
  - groups can be referenced for substitution!

Example: extract fields from unix date

```
$ date
Thu Jul 18 14:33:28 PDT 2019
```

- day of week: `[A-Z][a-z]{2}` (?# 1 cap,2 lower)
- month: `[A-Z][a-z]{2}` (?# 1 cap,2 lower)
- day: `0?[1-9]|[12]\d|3[01]`(?# 1-9,10-29, 0-31)
- time: `\d\d:\d\d:\d\d` (?# 2-digits hh:mm:ss)
- timezone: `[A-Z]{3}` (?# 3 capital letters)
- year: `\d{1,4}` (?# 1-4 digits)

re with grouping for date parsing

```
>>> pattern = r'\s+'.join([r'([A-Z][a-z]{2})', # day of week (Group 1)
                           r'([A-Z][a-z]{2})', # month       (Group 2)
                           r'([1-9]|[12]\d|3[01])', # day    (Group 3)
                           r'(\d\d):(\d\d):(\d\d)', # hh:mm:ss (4,5,6)
                           r'([A-Z]{3})',  # timezone         (Group 7)
                           r'(\d{1,4})'])  # year             (Group 8)
>>> pattern
'([A-Z][a-z]{2})\\s+([A-Z][a-z]{2})\\s+([1-9]|[12]\\d|3[01])\\s+(\\d\
\d):(\\d\\d):(\\d\\d)\\s+([A-Z]{3})\\s+(\\d{1,4})'
>>> m = re.fullmatch(pattern, 'Thu Jul 18 14:33:28 PDT 2019')
>>> m
<re.Match object; span=(0, 28), match='Thu Jul 18 14:33:28 PDT 2019'>
>>> m.group(1)
'Thu'
>>> m.group(8)
'2019'
>>> [m.group(i) for i in range(1,9)]
['Thu', 'Jul', '18', '14', '33', '28', 'PDT', '2019']
```

Substitution

- re.sub(pattern, replace, string)
  - pattern is a regular expression string
  - replace can reference groups as `\1` `\2` etc

- Different date formats
  - European: 15/08/2019 (day/month/year)
  - American: 08/15/2019 (month/day/year)
  - Asian: 2019/08/15   (year/month/day)
  - separators: `'-'`, `'.'`, `'/'` (e.g., 2019.08.15)

Example: Substituting the date separator

- Convert from `'-'` to `'/'`

```
>>> import re
>>> re.sub('-','/', 'today is 5-20-2019, tomorrow 5-21-2019')
'today is 2019/20/5, tomorrow 2019/21/5'
```

- Convert from European to Asian format

```
>>> import re
>>> re.sub(r'(\d{1,2})/(\d{1,2})/(\d{1,4})', r'\3/\2/\1',
           'today is 20/5/2019, tomorrow 21/5/2019')
'today is 2019/5/20, tomorrow 2019/5/21'
>>>
```

vim vs Python regular expression

meaning	vim	Python re
0 or more	*	*
1 or more	🚫	+
zero or one	🚫	?
set of characters	[ ]	[ ]
matches char not in set	[^]	[^]
grouping	\(\)	( )
n copies, *m* to *n* copies	🚫	{*n*} {*m*,*n*}
word boundary	\< \>	\b \b
beginning of line	^	^
end of line	$	$
beginning of string	🚫	\A
end of string	🚫	\Z

Graphical User Interface with Tkinter

- Tkinter = "Tk Interface"
  - Tk = "tool kit", a graphical user interface system
  - originally part of Tcl/Tk by John Ousterhout
  - tcl = "tool command language", similar to unix shell
- Tkinter provides "interface" to Tk
  - Python API call, translated into tcl
  - standard installation for Python
  - used to implement the IDLE environment

Widgets: GUI elements

- Button
- Canvas
- Checkbutton
- Entry
- Frame
- Label
- Listbox
- Menu
- Menubutton
- (Message) => use Label
- Radiobutton
- Scale
- Scrollbar
- Text
- Toplevel
- LabelFrame
- PanedWindow
- Spinbox

Basic concept of Tkinter program

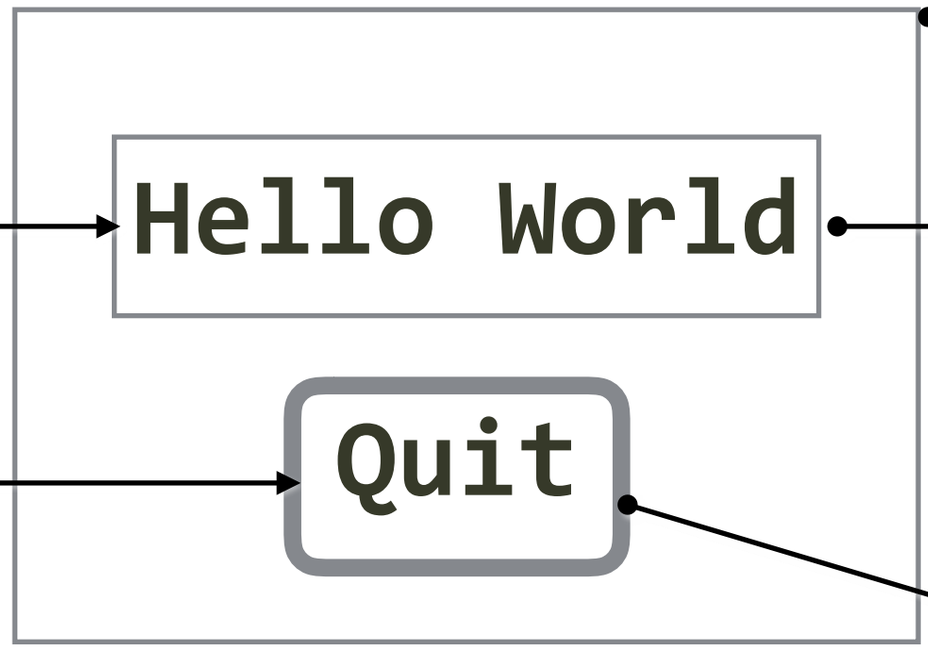
- Call tkinter.Tk() to instantiate the master
- Define or find callback function for widget
  - e.g., master.destroy is predefined for "Quit"
- Instantiate wedget
- Link widget with
  - parent widget
  - callback function
- tkinter.mainloop()

Example: "Hello world" v1

Frame widget f (parent = root)

Label widget l (parent = f)

Button widget b (parent = f)




```
import tkinter
root = tkinter.Tk()
```

```
f = tkinter.Frame(root)
f.pack()
```

```
l = tkinter.Label(f,
      text='Hello World')
l.pack()
```

```
b = tkinter.Button(f,
      text='Quit',
      command=root.destroy)
b.pack()
```

```
tkinter.mainloop()
```

Issue with Hello World v1

- Window default size a bit too small
  - solution: pass width and height to frame

```
f = tkinter.Frame(root, width=200, height=150)
f.pack_propagate(0)  # don't shrink
f.pack()
```

- Want label to be top, button be bottom
  - solution: pass side='top' or 'bottom' to widget

```
l = tkinter.Label(f, text='Hello World')
l.pack(side=tkinter.TOP)
```

```
b = tkinter.Button(f, text='Quit', command=root.destroy)
b.pack(side=tkinter.BOTTOM)
```

Source code for v2

```
import tkinter

root = tkinter.Tk()
f = tkinter.Frame(root, width=200, height=150)
f.pack_propagate(0) # don't shrink!
f.pack()

l = tkinter.Label(f, text='Hello World')
l.pack(side=tkinter.TOP)

b = tkinter.Button(f, text='Quit', command=root.destroy)
b.pack(side=tkinter.BOTTOM)

tkinter.mainloop()
```

Application: Calendar app

- Get today's date

```
import datetime
today = datetime.date.today()
```

- Instantiate calendar formatter

```
import calendar
cal = calendar.TextCalendar(6)
```

- Generate this month's calendar string
  - using today's year and month
  - to be used as Label's text

```
calstr = cal.formatmonth(today.year, today.month)
```

Issue with text calendar

- Assumes fixed-width font
  - Solution: Label constructor gets parameter font=('Courier', 12) or any other fixed width font
- Assumes left-justified text
  - Solution: Label constructor gets parameter justify=tkinter.LEFT

Complete source code for calendar app v1

```
import tkinter

root = tkinter.Tk()
f = tkinter.Frame(root, width=200, height=150)
f.pack_propagate(0) # don't shrink!
f.pack()

import datetime
today = datetime.date.today()
import calendar
cal = calendar.TextCalendar(6)
calstr = cal.formatmonth(today.year, today.month)

l = tkinter.Label(f, text=calstr, justify=tkinter.LEFT,
                  font=('Courier', 12))
l.pack(side=tkinter.TOP)

b = tkinter.Button(f, text='Quit', command=root.destroy)
b.pack(side=tkinter.BOTTOM)

tkinter.mainloop()
```

Issue with calendar v1

- displays only today's date
  - add buttons to go to next or previous month
- Solution for v2
  - instantiate two buttons (prevmonth, nextmonth)
  - define two callbacks to decrement and increment month, then refresh label content
  - link a StringVar to the Label (for calendar), so updating the StringVar will refresh Label

Calendar v2

- Define tkinter.StringVar for Label
  - pass to textvariable parameter of Label

```
calstr = tkinter.StringVar()
calstr.set(cal.formatmonth(current_year, current_month))
l = tkinter.Label(f, textvariable=calstr, justify=tkinter.LEFT,
                  font=('Courier', 12))
l.pack(side=tkinter.TOP)
```

- Define callbacks

```
def prev_month():  # called when user clicks prev button
    global current_year, current_month
    current_month -= 1
    if current_month == 0:
        current_month = 12
        current_year -= 1
    calstr.set(cal.formatmonth(current_year, current_month))
```

analogous for next_month

Calendar v2

- Add prev and next buttons
  - pass prev_month as command (callback) to Button's constructor call

```
pm = tkinter.Button(f, text='Prev', command=prev_month)
pm.pack(side=tkinter.LEFT)
```

```
nm = tkinter.Button(f, text='Next', command=next_month)
nm.pack(side=tkinter.RIGHT)
```

- That should do it! everything is connected

Source for calendar v2

```
import tkinter

root = tkinter.Tk()
f = tkinter.Frame(root, width=250, height=200)
f.pack_propagate(0) # don't shrink!
f.pack()

import datetime
today = datetime.date.today()
current_year, current_month = today.year, today.month
import calendar
cal = calendar.TextCalendar(6)

calstr = tkinter.StringVar()
calstr.set(cal.formatmonth(current_year, current_month))

l = tkinter.Label(f, textvariable=calstr, justify=tkinter.LEFT,
                  font=('Courier', 12))
l.pack(side=tkinter.TOP)

b = tkinter.Button(f, text='Quit', command=root.destroy)
b.pack(side=tkinter.BOTTOM)
```

Source for calendar v2 (cont'd)

```
def prev_month():
    global current_year, current_month
    current_month -= 1
    if current_month == 0:
        current_month = 12
        current_year -= 1
    calstr.set(cal.formatmonth(current_year, current_month))

def next_month():
    global current_year, current_month
    current_month += 1
    if current_month == 13:
        current_month = 1
        current_year += 1
    calstr.set(cal.formatmonth(current_year, current_month))

pm = tkinter.Button(f, text='Prev', command=prev_month)
pm.pack(side=tkinter.LEFT)

nm = tkinter.Button(f, text='Next', command=next_month)
nm.pack(side=tkinter.RIGHT)

tkinter.mainloop()
```

Issue with calendar v2

- Use of global variables
  - inevitable, especially in callbacks
  - could be better packaged (e.g., as one object)
- Multiple copies of similar calls
  - prev_month() and next_month() are similar, could be merged into the same function with a parameter to indicate incr/decr
  - calstr.set(cal.formatmonth(current_year, current_month)) appears multiple times, should be restructured
- Layout is not neat
  - button too close to edge, not aligned

Code refactoring: merging prev_month() and next_month()

- Idea: pass `operator.add` or `operator.sub` to indicate increment or decrement!

```
def shift_month(add_or_sub):
    global current_year, current_month
    current_month = add_or_sub(current_month, 1)
    if current_month == 0:
        current_month = 12
        current_year -= 1
    elif current_month == 13:
        current_month = 1
        current_year += 1
    calstr.set(cal.formatmonth(current_year, current_month))
```

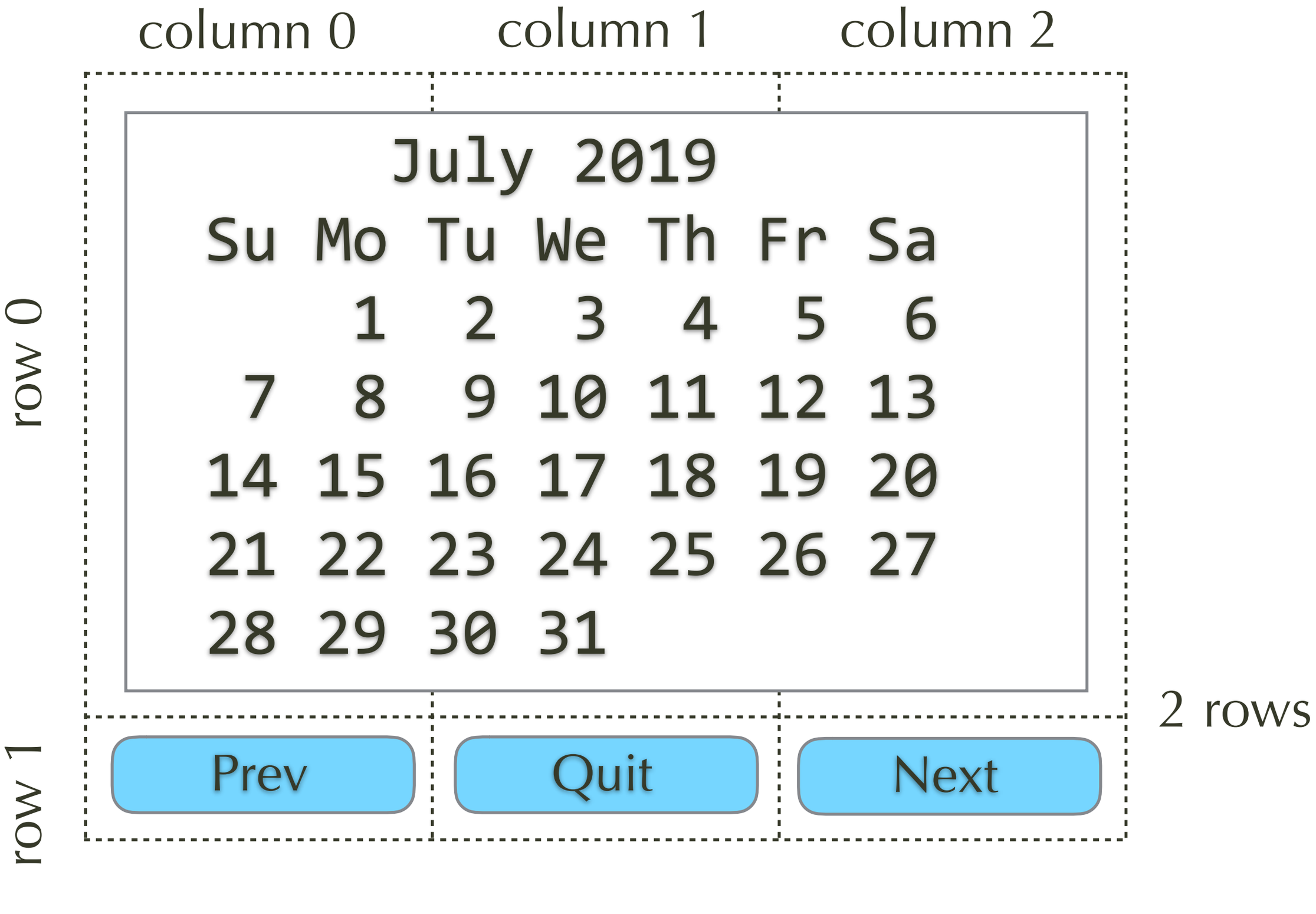
- Pass lambda as callback

```
import operator
next_month = lambda : shift_month(operator.add)
prev_month = lambda : shift_month(operator.sub)
```

Grid layout manager

- Tkinter provides three layout managers:
  - pack - simplest, but may not be aligned
  - grid - expandable table, good for most purpose
  - place - exact placement, can tweak exact look, but tedious
- Use grid layout to align widgets in table
  - row, column, rowspan, columnspan
  - add border width to Frame for better look

Grid layout for Calendar



instead of pack(), call grid() on each widget

- Frame can have border width

```
f = tkinter.Frame(root, borderwidth=5)
```

- Frame also need to be put in root's grid

```
f.grid(row=0, column=0)
```

- Label takes entire row 0, column span = 3

```
l.grid(row=0, column=0, columnspan=3) # instead of l.pack(TOP)
```

- Each button occupies a column in row 1

```
pm.grid(row=1, column=0)          # instead of pm.pack(LEFT)
b.grid(row=1, column=1)           # instead of b.pack(BOTTOM)
nm.grid(row=1, column=2)          # instead of nm.pack(RIGHT)
```

Source for calendar v3

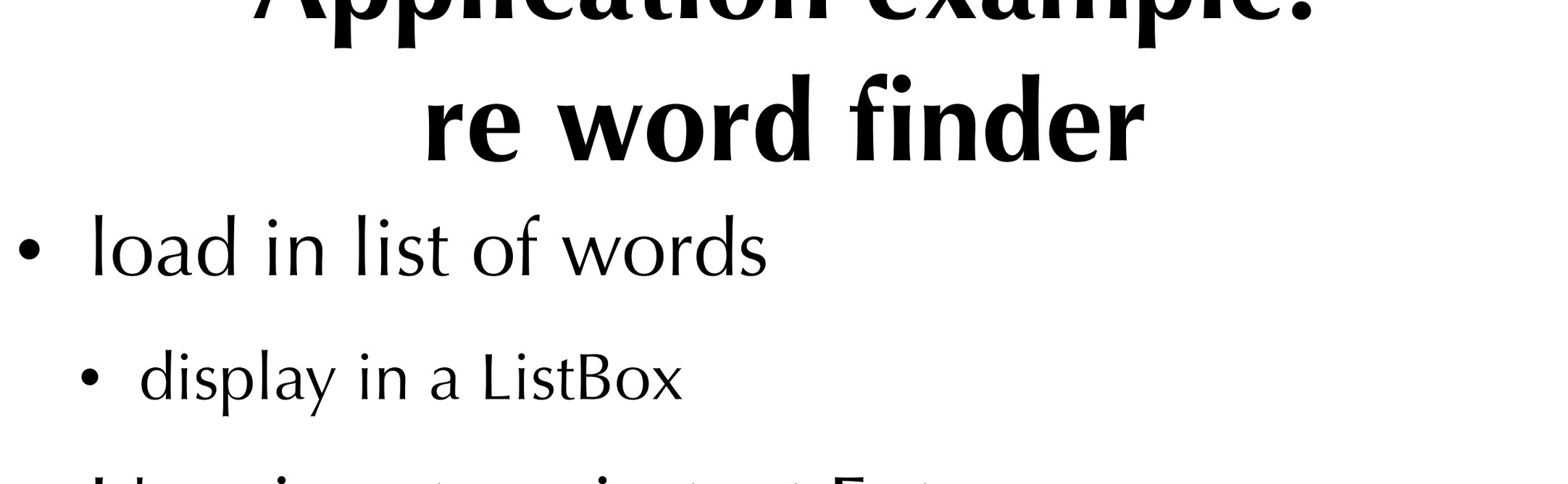
```
import tkinter
from operator import add, sub
root = tkinter.Tk()
f = tkinter.Frame(root, borderwidth=5)
import datetime
today = datetime.date.today()
current_year, current_month = today.year, today.month
import calendar
cal = calendar.TextCalendar(6)
calstr = tkinter.StringVar()
calstr.set(cal.formatmonth(current_year, current_month))
# code for shift_month() omitted here; see code refactoring slide
l = tkinter.Label(f, textvariable=calstr, justify=tkinter.LEFT,
                  font=('Courier', 12))
b = tkinter.Button(f, text='Quit', command=root.destroy)
nm = tkinter.Button(f, text='Next', command=lambda:shift_month(add))
pm = tkinter.Button(f, text='Prev', command=lambda:shift_month(sub))

f.grid(row=0, column=0)
l.grid(row=0, column=0, columnspan=3)
[w.grid(row=1, column=i) for i,w in enumerate([pm, b, nm])]
```

More widgets: Entry and Listbox

- `Entry`: one-line text input
  - can use StringVar() to access value
- `ListBox`: text list for browsing and picking
  - `insert()` a string into a position
  - `delete()` string at index or range
  - `curselection()` = selected items

Application example: re word finder

- load in list of words
  - display in a ListBox
- User inputs re in text Entry
- click Find to display matched words



Technical approach

- create Frame and widgets
  - Label, Entry, Button, Listbox
- read dictionary words into a list
  - /usr/share/dict/words or /usr/dict/words on macOS, Linux
  - http://ftp.gnu.org/gnu/aspell/dict/ for other languages
- define callback for re search over list
  - get regular expression from Entry's StringVar
  - build list of matched words by calling re.search on each word
  - delete all previous items and add new items to Listbox

User interface for word finder

```
import tkinter
root = tkinter.Tk()
f = tkinter.Frame(root, borderwidth=5)
f.grid(row=0, column=0)
```

```
pat_label = tkinter.Label(f,
                  text='Pattern')
pat_label.grid(row=0, column=0)
```

```
find_btn = tkinter.Button(f,
    text='Find', command=apply_reg_expr)
find_btn.grid(row=0, column=2)
```


```
pat_str = tkinter.StringVar()
pat_entry = tkinter.Entry(f,
          textvariable=pat_str)
pat_entry.grid(row=0, column=1)
```

```
result_box = tkinter.Listbox(f)
result_box.grid(row=1, column=0, columnspan=3)
```

code for callback for search

```
def apply_reg_expr():  # callback function
    import re
    regexp = pat_str.get()  # get string value from Entry
    if len(regexp) == 0: return
    # now try matching
    matched_words = [w for w in words if re.search(regexp, w) != None]
    # delete old content in Listbox
    result_box.delete(0, tkinter.END)
    for i, w in enumerate(matched_words):
        result_box.insert(i, w)
```

- Assumption
  - global `words` contains list of words (from file)

Code for reading words from file

```
try:
    fh = open('/usr/share/dict/words')
    words = list(map(lambda x: x[:-1] if x[-1]=='\n' else x,
                     fh.readlines()))
    fh.close()
except:
    print('cannot open file')
    root.destroy()
```

- `fh.readlines()` => list of words, one per line
  - issue: includes '\n'! want to eliminate it
- `lambda x: [x:-1] if x[-1]=='\n' else x`
  - check if x[-1] (last char is '\n', if so, take x[:-1] => word w/out \n
- `list()`
  - convert `map(`of every word to w/out \n) from iterator into a list

Patterns to try

- words with six o's
  - `(.*?o){6}`
- words that duplicates a 3-letter pattern
  - `(.{3}).*\1`
- 7-letter conundrum
  - `^(.)(.)(.).\3\2\1$`
- words with at least five consecutive vowel letters
  - `[aeiouAEIOU]{5}`

threads

- What is a thread?
  - a unit of execution within a program
  - threads => can be at different parts of a program!
- so far, our programs are single-threaded
  - you can be in one place at any given moment
- threading module
  - allows you to create Threads to run a function "in parallel" to other threads

Why use threads?

- Example: mainloop in tkinter
  - tkinter: last line in program was f.mainloop()
    => tkinter code is running the main loop!
  - while in mainloop, none of your code gets called unless user clicks a button, types words, etc.
    => you only get callbacks!
- How to build a clock?
  - single threaded => clock won't get updated!
  - Solution: create a thread to update clock string once/sec

code for updating clock

```
import threading, time
def update_clock():
    while not quit:
        now = datetime.datetime.now()
        clockstr.set(f'''Date: {now.year}/{now.month}/{now.day}
Time: {now.hour:02d}:{now.minute:02d}:{now.second:02d}''')
        time.sleep(1)
```

- assumption: global variable named `quit`
  - initialized to `False` to start, set to `True` when GUI quits
- assumption: `clockstr` is a `StringVar`
  - set as `textvariable` parameter for a `Label` to display date and time as text
- `time.sleep(1)` sleeps (the thread) for 1 second

code for creating thread to run update_clock

```
th = threading.Thread(target=update_clock)
th.start()
```

- `th = threading.Thread(target=update_clock)`
  - create a Thread object `th` will run target function `update_clock`
  - thread does not actually run until `th.start()`
- `th.start()`
  - main program continues running, but the second thread also gets opportunities to run concurrently

Whole code for clock using threads

```
import tkinter, datetime, time, threading
quit = False

def update_clock():
    # see source code on previous slide

root = tkinter.Tk()
f = tkinter.Frame(root, borderwidth=5)
root.title('Clock')   # set our own title words for root window
clockstr = tkinter.StringVar()
l = tkinter.Label(f, textvariable=clockstr)
f.grid(row=0, column=0)
l.grid(row=0, column=0)

th = threading.Thread(target=update_clock)
th.start()
f.mainloop()
quit = True  # mainloop() returns here after UI closes, so we want
             # set quit=True to make update_clock thread exit loop
```