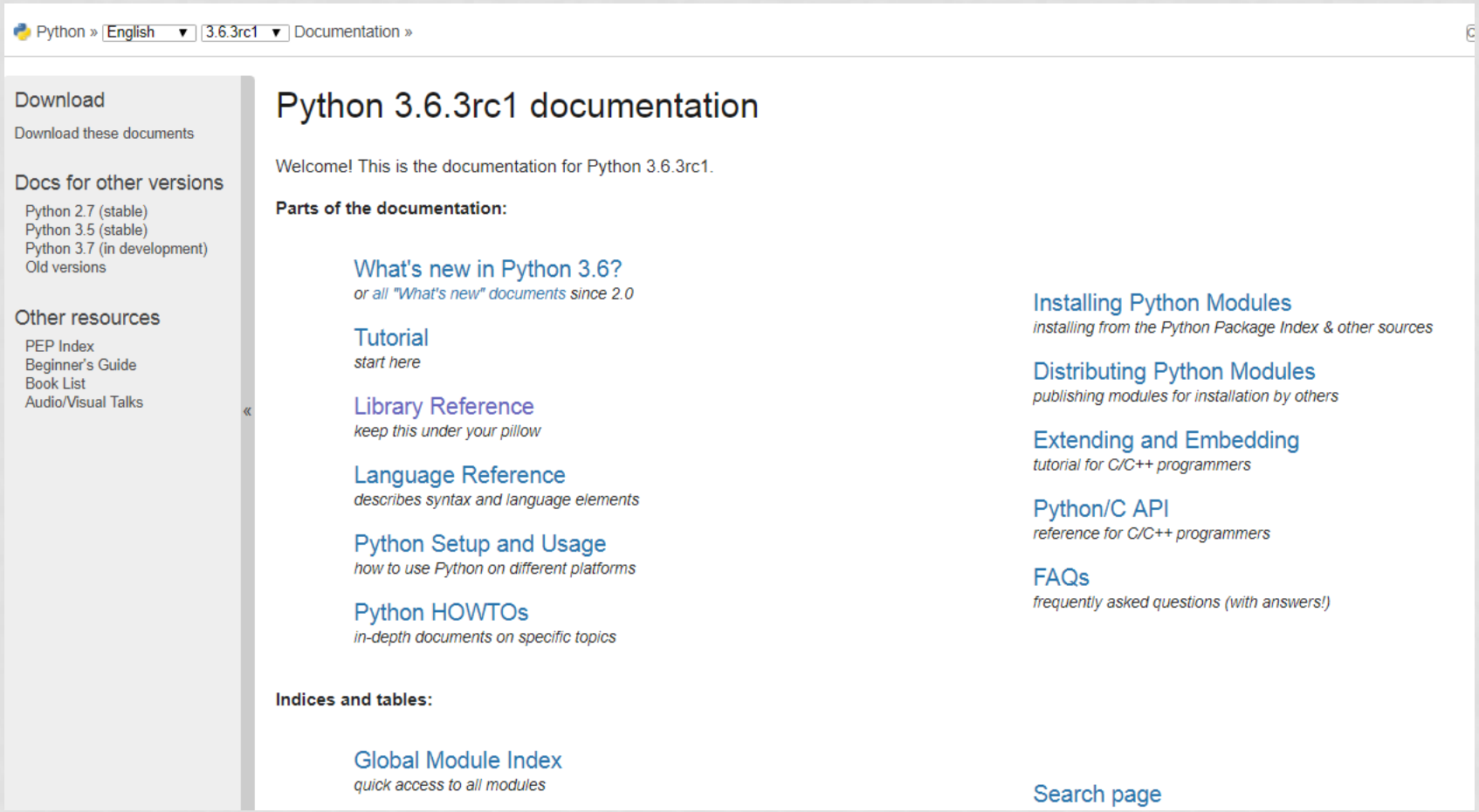
LECTURES WEEK 4

AGENDA

week	subject	book	week	subject	book
1	Python features	1	3	any & all	19
	running Python	1		range, zip & enumerate	12
	dynamic binding	2		higher-order functions	16
	Python statements	1		classes and OOP	15..18
	printing stuff	2,3		exceptions	14
	Python types	1		assert	16
	numbers	1		file access	14
	strings	8		working with CSV and JSON	-
	control statements	7		coding style	-
	lists	10			
2	tuples	12	4	case: word histogram	13
	dictionaries	11		recursion	5
	sets	19		case: solving Numbrix	-
	functions	6		PySerial	-
	scope/visibility	11		tkinter GUI-toolkit	-
	comprehension	19		web-programming	
	modules and packages	14			

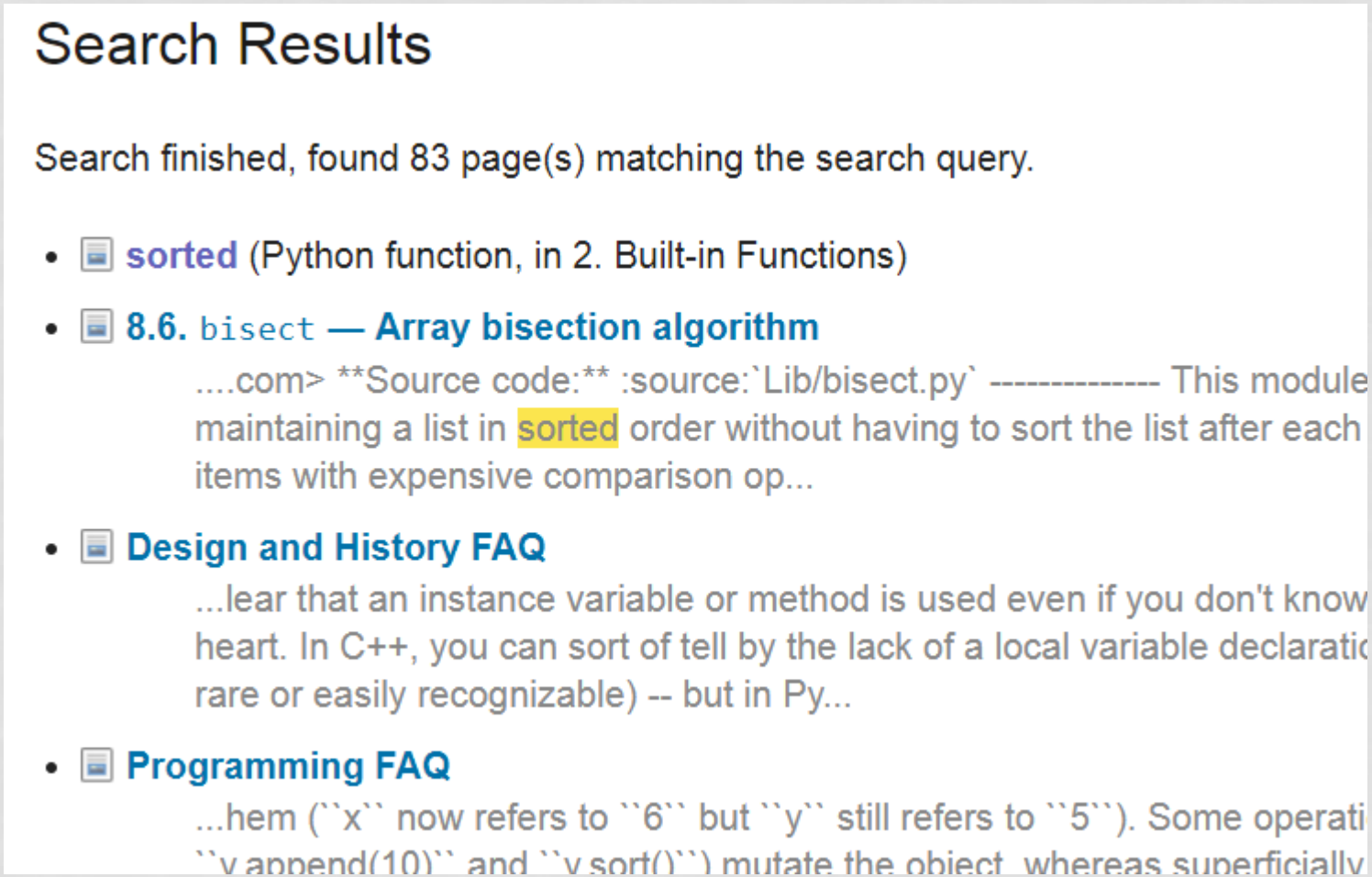
AGENDA

- **Python documentation**
- case: word histogram
- recursion
- case: solving Numbrix
- PySerial
- tkinter GUI-toolkit
- web-programming


Python » English 3.6.3rc1 Documentation »
Download
Download these documents
Docs for other versions
Python 2.7 (stable)
Python 3.5 (stable)
Python 3.7 (in development)
Old versions
Other resources
PEP Index
Beginner's Guide
Book List
Audio/Visual Talks
Python 3.6.3rc1 documentation
Welcome! This is the documentation for Python 3.6.3rc1.
Parts of the documentation:
What's new in Python 3.6?
or all "What's new" documents since 2.0
Tutorial
start here
Library Reference
keep this under your pillow
Language Reference
describes syntax and language elements
Python Setup and Usage
how to use Python on different platforms
Python HOWTOs
in-depth documents on specific topics
Installing Python Modules
installing from the Python Package Index & other sources
Distributing Python Modules
publishing modules for installation by others
Extending and Embedding
tutorial for C/C++ programmers
Python/C API
reference for C/C++ programmers
FAQs
frequently asked questions (with answers!)
Indices and tables:
Global Module Index
quick access to all modules
Search page

PYTHON DOCUMENTATION

PYTHON DOCUMENTATION



Search Results

Search finished, found 83 page(s) matching the search query.

- **sorted** (Python function, in 2. Built-in Functions)
- **8.6. `bisect` — Array bisection algorithm**

  ....com> **Source code:** :source:`Lib/bisect.py` ---------- This module maintaining a list in sorted order without having to sort the list after each items with expensive comparison op...
- **Design and History FAQ**

  ...lear that an instance variable or method is used even if you don't know heart. In C++, you can sort of tell by the lack of a local variable declaratio rare or easily recognizable) -- but in Py...
- **Programming FAQ**

  ...hem (``x`` now refers to ``6`` but ``y`` still refers to ``5``). Some operati ``y.append(10)`` and ``y.sort()``) mutate the object, whereas superficially

PYTHON DOCUMENTATION

sorted(*iterable*, *, *key=None*, *reverse=False*)

Return a new sorted list from the items in *iterable*.

Has two optional arguments which must be specified as keyword arguments.

key specifies a function of one argument that is used to extract a comparison key from each list element: `key=str.lower`. The default value is `None` (compare the elements directly).

reverse is a boolean value. If set to `True`, then the list elements are sorted as if each comparison were reversed.

Use `functools.cmp_to_key()` to convert an old-style *cmp* function to a *key* function.

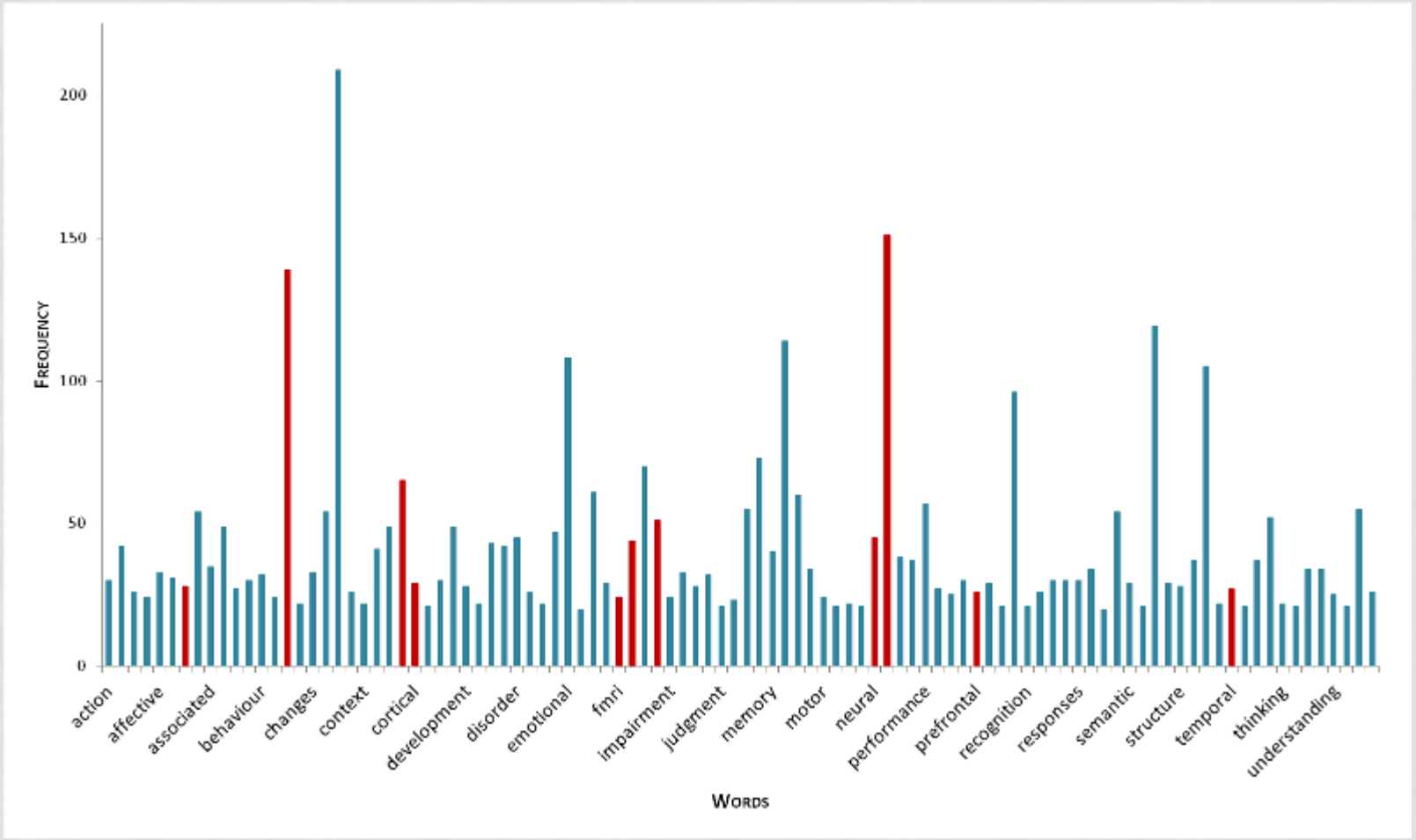
The built-in `sorted()` function is guaranteed to be stable. A sort is stable if it guarantees not to change the relative order of elements that compare equal — this is helpful for sorting in multiple passes (for example, sort by department, then by salary grade).

For sorting examples and a brief sorting tutorial, see Sorting HOW TO.

AGENDA

- Python documentation
- **case: word histogram**
- recursion
- case: solving Numbrix
- PySerial
- tkinter GUI-toolkit
- web-programming

WORD HISTOGRAM



WORDS HISTOGRAM

- write a program that reads a file and builds a histogram (top 20) of the words in the file

- what do we need?
  - data structures & algorithms (steps)
- how to represent a histogram (model)?
  - dict keys:words, values: frequency
- algorithm?
  - read a line
  - split line into words
  - increase counts in histogram

COLLECTIONS.COUNTER

```
1  from collections import Counter
2
3  counter = Counter('abracadabra')
4  print(counter)
5  counter.update('aaaaazzz')
6  print(counter)
7  print(counter.most_common(2))
8
```

```
Counter({'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1})
Counter({'a': 10, 'z': 3, 'b': 2, 'r': 2, 'c': 1, 'd': 1})
[('a', 10), ('z', 3)]
```

SOLUTION

```
 1  from collections import Counter
 2
 3  histo = Counter()
 4
 5  with open('emma.txt', 'r') as f:
 6      for line in f:
 7          for word in line.strip().split():
 8              word = word.lower()
 9              histo[word] = histo.get(word, 0) + 1
10
11  for word, count in histo.most_common(20):
12      print(count, word)
```

MORE PYTHONIC

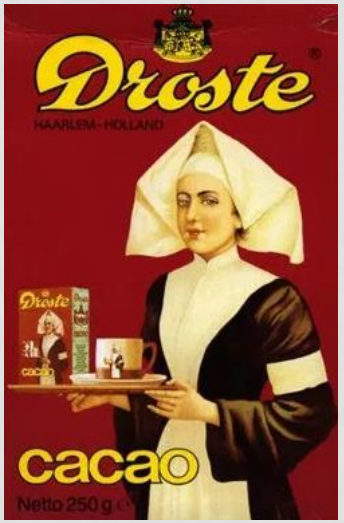
```
 1  from collections import Counter
 2
 3  with open('emma.txt', 'r') as f:
 4      histo = Counter(word.lower()
 5                      for line in f
 6                      for word in line.strip().split()
 7                      if word)
 8
 9  for word, count in histo.most_common(20):
10      print(count, word)
```

AGENDA

- Python documentation
- case: word histogram
- **recursion**
- case: solving Numbrix
- PySerial
- tkinter GUI-toolkit
- web-programming

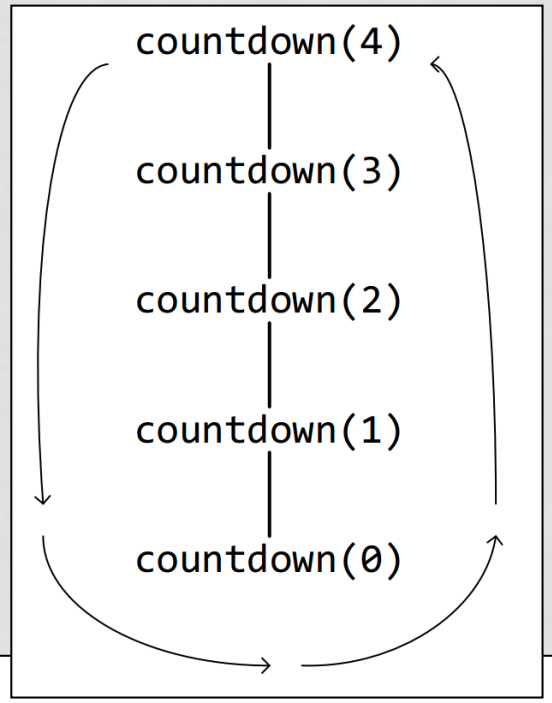
RECURSION

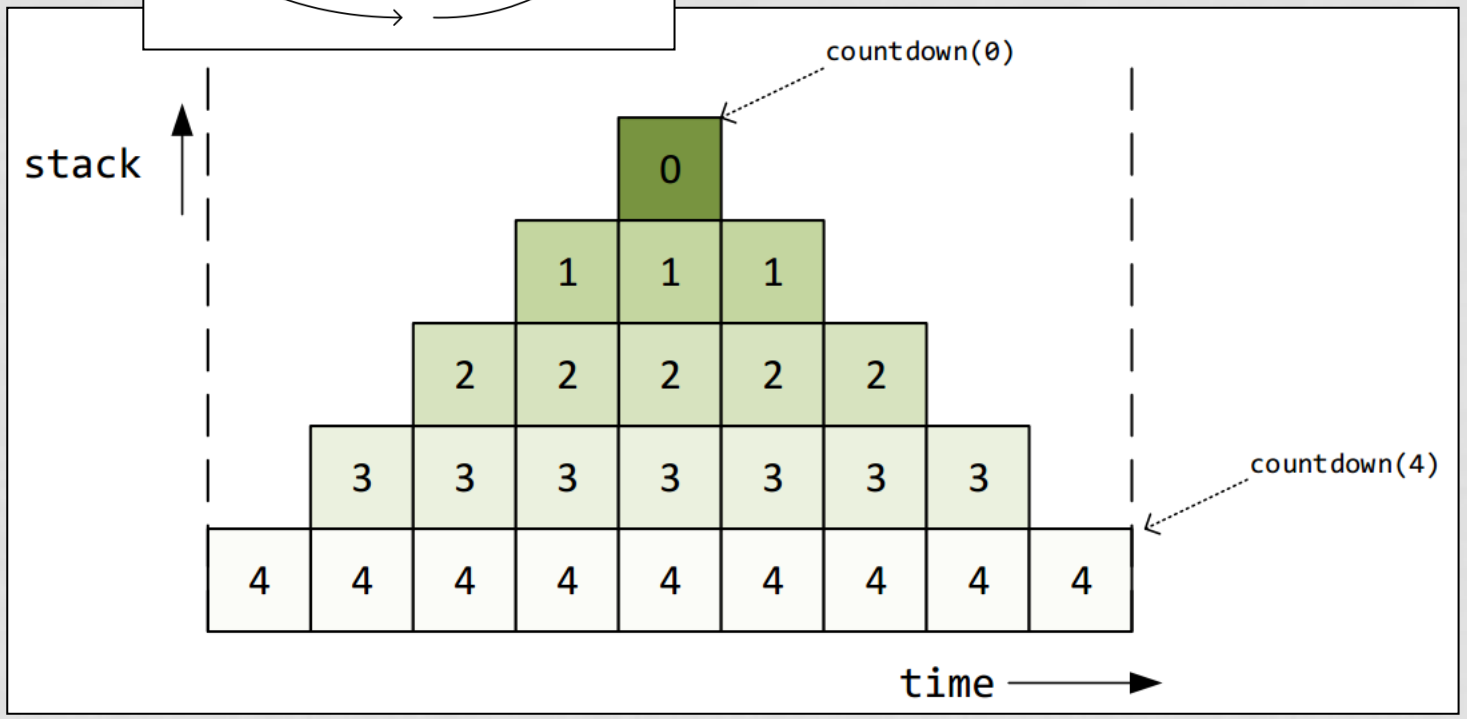
- function may call another function, but it may also call itself: recursion
- recursion can be very powerful, e.g. when the solution to a problem depends on solutions to smaller instances of the same problem
- every recursive call reduces the original problem, bringing it closer to a base case
- some classic examples:
  - factorial, Fibonacci, gcd, towers of Hanoi
  - list- and tree-traversal



A SIMPLE EXAMPLE

```
1  def countdown(n):
2      if n == 0:
3          print('We\'re done!')
4      else:
5          print(n)
6          countdown(n-1)
7
8  countdown(10)
```


countdown(4)
countdown(3)
countdown(2)
countdown(1)
countdown(0)


countdown(0)
stack
0
1 1 1
2 2 2 2 2
3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4
countdown(4)
time

WHAT'S THE DIFFERENCE ?

```
1  def countdown(n):
2      if n == 0:
3          print('We\'re done!')
4      else:
5          print(n)
6          countdown(n-1)
7
8  countdown(10)
```

```
10
9
8
7
6
5
4
3
2
1
We're done!
```

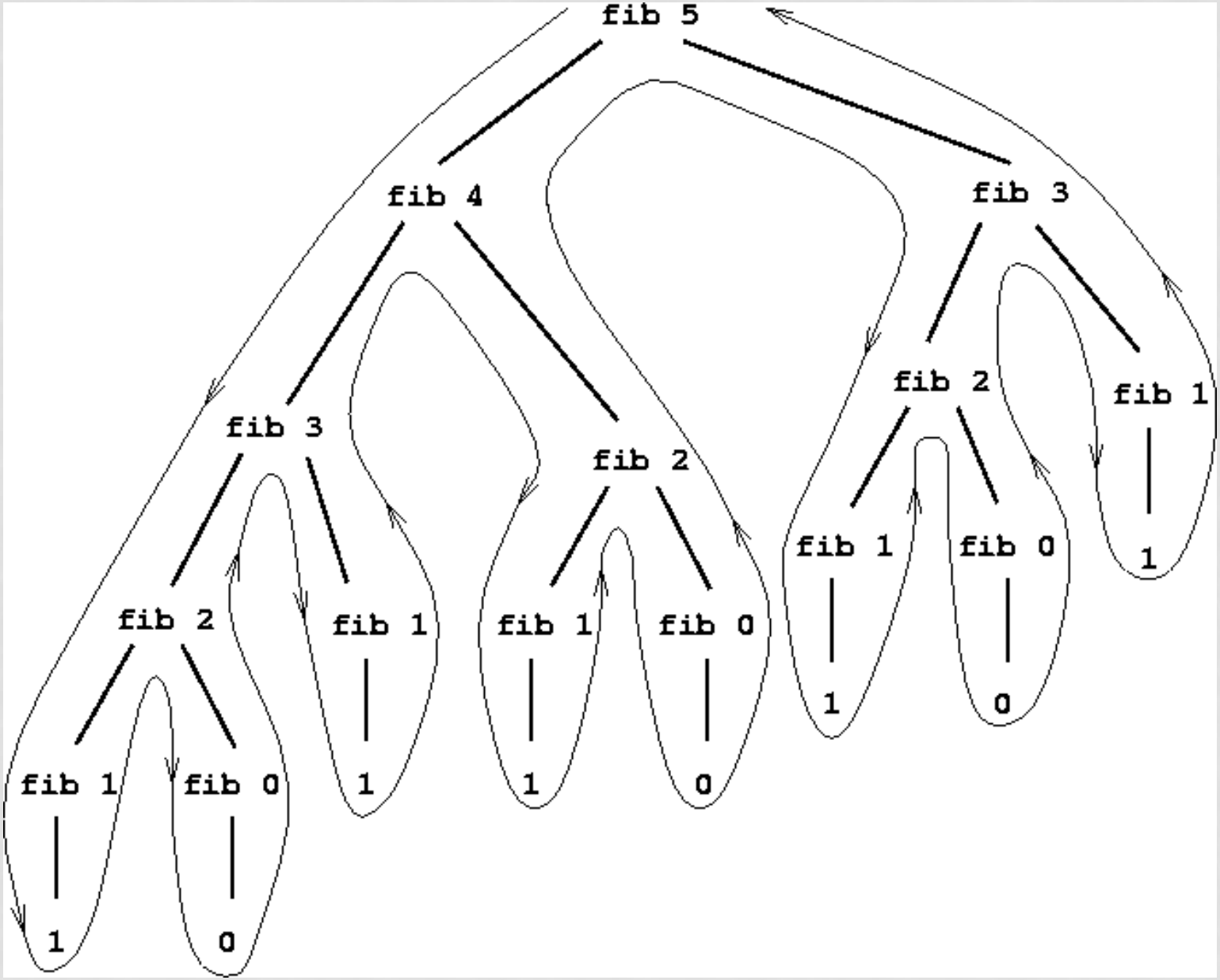
```
1  def countdown(n):
2      if n == 0:
3          print('We\'re done!')
4      else:
5          countdown(n-1)
6          print(n)
7
8  countdown(10)
```

```
We're done!
1
2
3
4
5
6
7
8
9
10
```

FIBONACCI SEQUENCE

The series:	0	1	1	2	3	5	8	13	21	34	55	89	…
indexes:	0	1	2	3	4	5	6	7	8	9	10	11	

```
1  def fib(n):
2      if n==0:
3          return 0
4      elif n==1:
5          return 1
6      else:
7          return fib(n-1) + fib(n-2)
8
9  print(fib(5))
```


fib 5
fib 4
fib 3
fib 3
fib 2
fib 2
fib 1
fib 2
fib 1
fib 0
fib 1
fib 0
fib 1
fib 0
1
0
1
1
0
1
0
1

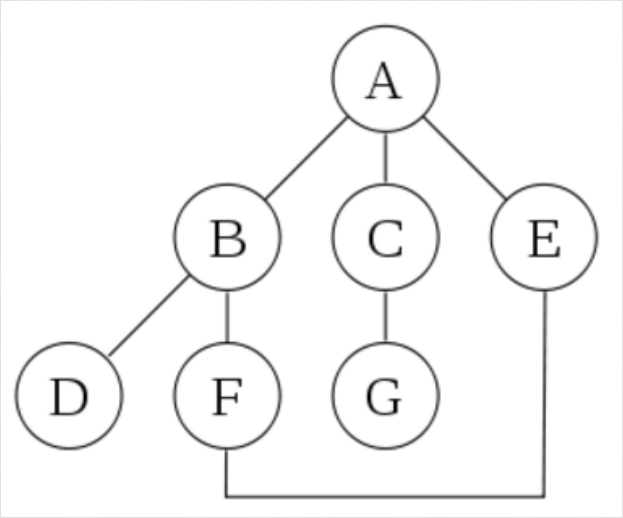
RECURSION

- the function handles different cases using an if-then-else
- there must be a stop-criterion to stop recursion

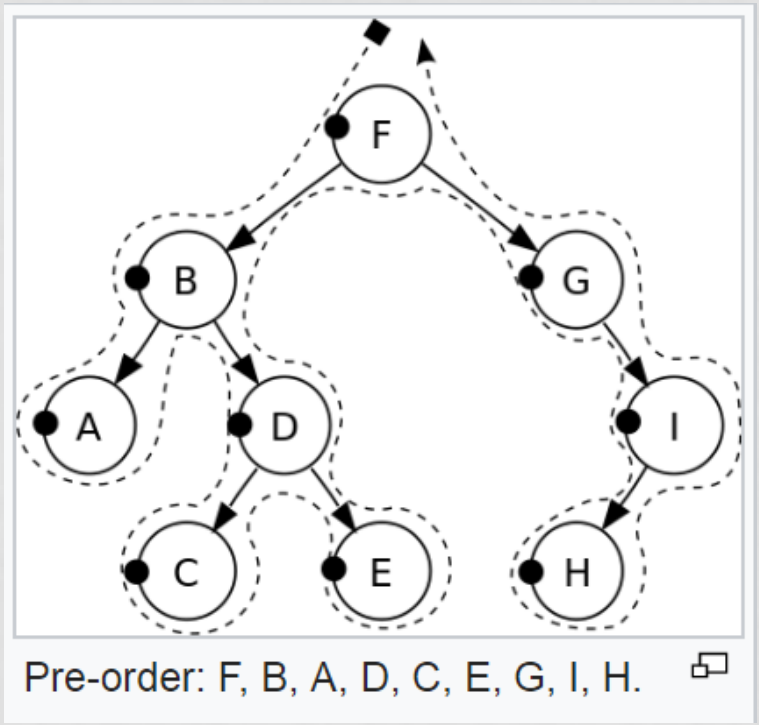
- recursion can always be implemented using iteration (that's what the CPU does)
- recursion may result in more elegant solutions than if using iteration

TREE-TRAVERSAL

- tree traversal is the process of visiting each node in the tree exactly once
- tree vs graph: a tree has no cycles, therefore no need need to track nodes that have already been visited
- a binary tree each node has at most two children
- the depth-first (DFS) traversal of a binary tree, the current node is visited first, then recursively the left subtree of the current node, and recursively the right subtree of the current node



graph with one cycle



a binary tree

DFS TREE (NO CYCLES)

```
1  def dfs_tree(node):
2      # print nodes pre-order
3
4      # first print the data of node
5      print(node.value)
6
7      if node_has_a_left_child:
8          # recursively traverse left subtree
9          print_preorder(node.left)
10
11     if node_has_a_right_child:
12         # recursively traverse right subtree
13         print_preorder(node.right)
```

DFS GRAPH (WITH CYCLES)

```
1  visited = set()
2
3  def dfs(graph, node):
4      visited.add(node)
5      print(node)
6      for all neighbors of node:
7          if neigbor not in visited
8              dfs(graph, neigbor)
```

AGENDA

- Python documentation
- case: word histogram
- recursion
- **case: solving Numbrix**
- PySerial
- tkinter GUI-toolkit
- web-programming

NUMBRIX PUZZLE

- a grid of cells is given, an n×n board
- number 1 and the highest number n×n are given, and some addition numbers (clues)
- the goal of Numbrix is to find the path from 1 to the highest number that covers the grid entirely
- numbers on a (Hamiltonian) path must be adjacent to each other vertically or horizontally (not diagonally)

NUMBRIX PUZZLE

- the puzzle is supposed to have a unique solution
- is a Constraint Satisfaction Problem (CSP), like Sudoku
- see Hidato on Wiki and Alex Bellos

7		3		1		59		81
			33	34	57			
9		31				63		79
	29						65	
11	12			39			66	77
	13						67	
15		23				69		75
			43	42	49			
19		21		45		47		73

A SIMPLE 3X3 NUMBRIX PUZZLE

1		
	5	
		9

1, 5, and 9 are called clues

A SOLUTION

1	2	3
6	5	4
7	8	9

how many solutions are there?

WHAT DO WE NEED?

- data structures & algorithms

- representation of the grid (=model)
- helper function that returns neighbors of a cell
- helper function that displays the board
- function that tries to find a solution: a path, given a list of clues (1, n×n, and other values)

REPRESENTATION OF THE GRID

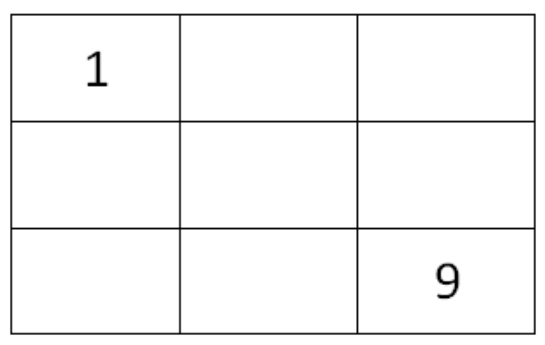
- string or list
- 1 or 2 dimensions?
- how to represent empty cells?
- how to find neighbors easily?

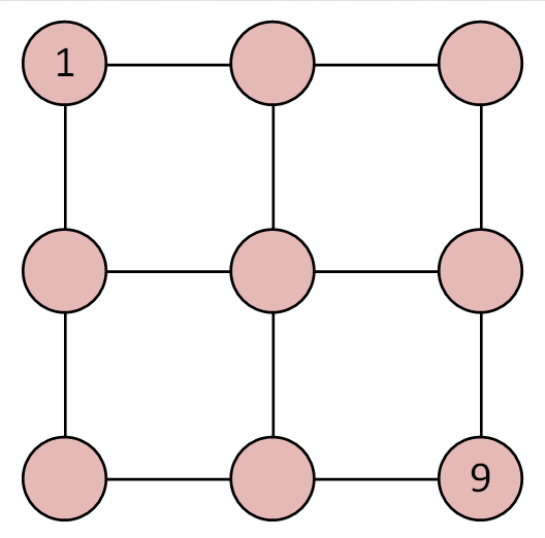
- since board is n×n, a simple 1-dimensional list will do
- empty cells get value = 0
- for neighbors calculate % n

HELPER FUNCTIONS

- function that displays the board
  - input is a list of numbers
  - return a string representation for printing
- function that returns neighbors of a cell
  - needs some thinking
  - row = list index // n
  - column = list index % n

HOW MANY PATHS?

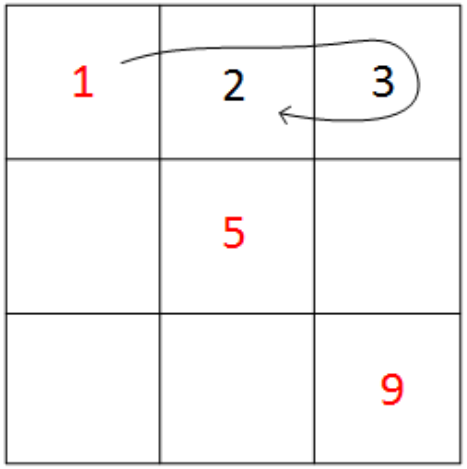




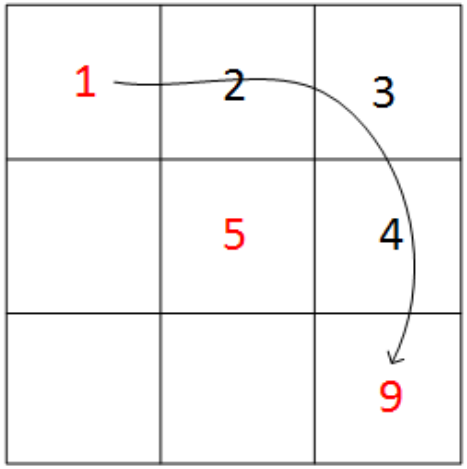
STRATEGY

- traverse all possible paths using recursion
- we can represent all possible paths as a tree, root node = 1 (cycles are not allowed)
- depth-first search: start at the root and explore each path as far as possible before backtracking
- as we follow a path, we fill the cells with a step counter
- if we backtrack, then have to remove step counter values

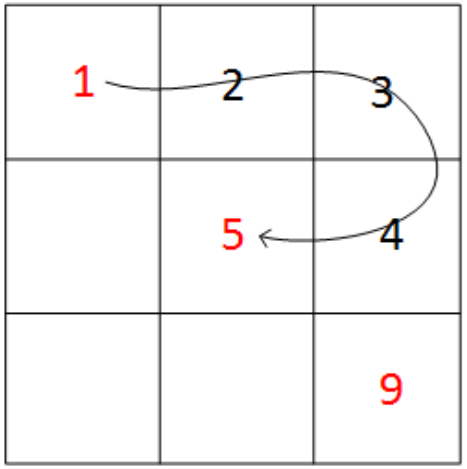
STRATEGY



invalid path



invalid path



valid path

STRATEGY

- which paths can be part of a solution?
  - if cell is empty & step count != next clue value *
  - if cell is not empty & step count = cell value
- which paths are not part of a solution?
  - if cell is empty & step count == next clue value *
    - clue doesn't match the step count
  - if cell is not empty & step count != cell value
    - we have been here before, it's a cycle

- * in list of clues

STRATEGY

```
1  # using DFS to traverse all paths
2  def solve(position, stepcount, next_clue_in_list):
3
4      can this be a valid path?
5      if yes: update the board and continue
6      if no: return
7
8      if found the last clue nxn
9          # we're done, found a solution
10         display the path
11         return
12     for n in neighbors(position):
13         # try all neigbours, increase the stepcount
14         solve(n, stepcount+1, next_clue_in_list)
15
16 solve(first_position, 1, next_clue_in_list)
```

AGENDA

- Python documentation
- case: word histogram
- recursion
- case: solving Numbrix
- **PySerial**
- tkinter GUI-toolkit
- web-programming

INSTALL PYSERIAL

- pip install pyserial

PYSERIAL API

class `serial.`**`Serial`**

`__init__`(*port=None*, *baudrate=9600*, *bytesize=EIGHTBITS*, *parity=PARITY_NONE*, *stopbits=STOPBITS_ONE*, *timeout=None*, *xonxoff=False*, *rtscts=False*, *write_timeout=None*, *dsrdtr=False*, *inter_byte_timeout=None*)

Parameters:
- **port** – Device name or `None`.
- **baudrate** (*int*) – Baud rate such as 9600 or 115200 etc.
- **bytesize** – Number of data bits. Possible values: `FIVEBITS`, `SIXBITS`, `SEVENBITS`, `EIGHTBITS`
- **parity** – Enable parity checking. Possible values: `PARITY_NONE`, `PARITY_EVEN`, `PARITY_ODD` `PARITY_MARK`, `PARITY_SPACE`
- **stopbits** – Number of stop bits. Possible values: `STOPBITS_ONE`, `STOPBITS_ONE_POINT_FIVE`, `STOPBITS_TWO`
- **timeout** (*float*) – Set a read timeout value.
- **xonxoff** (*bool*) – Enable software flow control.
- **rtscts** (*bool*) – Enable hardware (RTS/CTS) flow control.
- **dsrdtr** (*bool*) – Enable hardware (DSR/DTR) flow control.
- **write_timeout** (*float*) – Set a write timeout value.
- **inter_byte_timeout** (*float*) – Inter-character timeout, `None` to disable (default).

Raises:
- **ValueError** – Will be raised when parameter are out of range, e.g. baud rate, data bits.
- **SerialException** – In case the device can not be found or can not be configured.

The port is immediately opened on object creation, when a *port* is given. It is not opened when *port* is `None` and a successive call to `open()` is required.

`read`(*size=1*) ¶

Parameters: **size** – Number of bytes to read.
Returns: Bytes read from the port.
Return type: bytes

Read *size* bytes from the serial port. If a timeout is set it may return less characters as requested. With no timeout it will block until the requested number of bytes is read.

Changed in version 2.5: Returns an instance of `bytes` when available (Python 2.6 and newer) and `str` otherwise.

EXAMPLE

```
1  import serial
2  # open serial port at 19k2 (default = 8 data bits, 1 stop bit, no parity)
3  ser = serial.Serial('COM3', 19200)
4  print(ser)           # check which port was really used
5  while True:
6      s = ser.read()  # read single (raw) byte
7      print(s.hex())  # print as hex instead of b'...
```

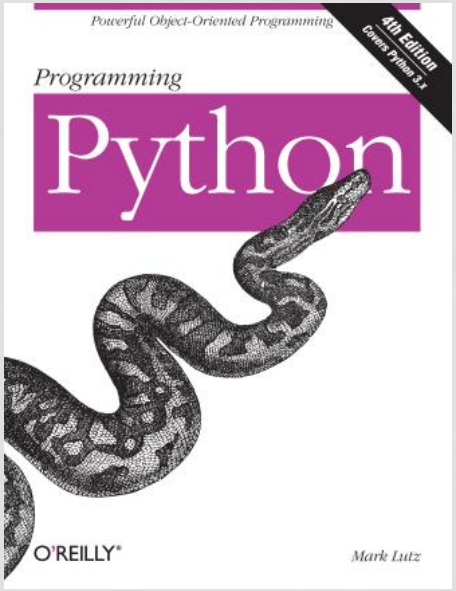
```
D:\arduino>python serial_test.py
Serial<id=0x1bf0b63048, open=True>(port='COM3', baudrate=19200, bytesize=8, parity='N', stopbits=1,
timeout=None, xonxoff=False, rtscts=False, dsrdtr=False)
33
77
bb
33
77
bb
33
77
```

AGENDA

- Python documentation
- case: word histogram
- recursion
- case: solving Numbrix
- PySerial
- **tkinter GUI-toolkit**
- web-programming

RESOURCES

- http://effbot.org/tkinterbook/
- http://www.tkdocs.com/tutorial/index.html
- https://www.python-course.eu/python_tkinter.php
- https://docs.python.org/3.6/library/tk.html



TKINTER

- is part of the standard library
- a lightweight GUI toolkit
- 25 basic widgets, various dialogs and other tools
- Tkinter is the Python interface to Tk ('tickle')
  - Tk is a GUI toolkit part of the Tcl distribution
    - Tcl is a scripting language (Tool Command Language)
  - a set of wrappers that implement the Tk widgets as Python classes

OTHER PYTHON GUI TOOLKITS

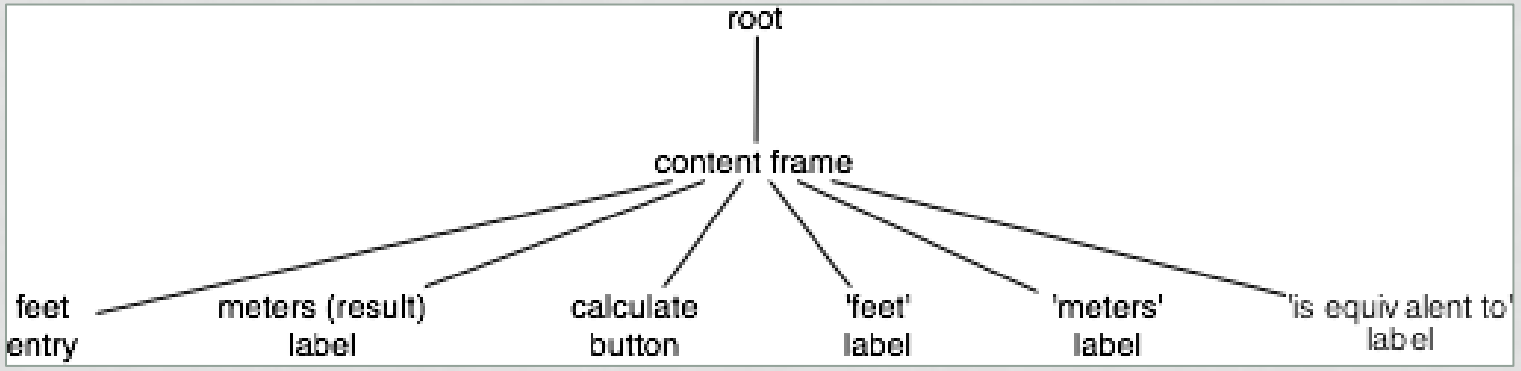
- ttk and tix
  - extension toolkits, adding new widgets or same widgets but better look and feel
  - both are modules in tkinter : `from tkinter import ttk`

OTHER PYTHON GUI TOOLKITS

- wxPython
  - Python bindings to wxWidgets
  - large toolkit originally to be used in C++
  - portable to UNIX, Windows and Mac
- PyQt/PySide
  - Python bindings to Qt
  - very similar, PySide has friendlier licensing scheme
  - originally to be used in C++
  - portable to UNIX, Windows and Mac

WIDGETS

- frames, buttons, labels, checkboxes, text area's
- widgets are objects, instances of classes
- all widgets are nodes in a widget-tree
  - you can have a button in a frame in another frame within the root window
- container widgets: top-level window and frames
- when creating a widget, you must pass its parent as a parameter to the widget creation function



GEOMETRY MANAGEMENT

- tasks for a geometry manager:
  - determining the size and position of components
  - arrange widgets on the screen
  - register widgets with the underlying windowing system
  - manage the display of widgets on the screen
- TK has 3 geometry managers: pack, grid and place

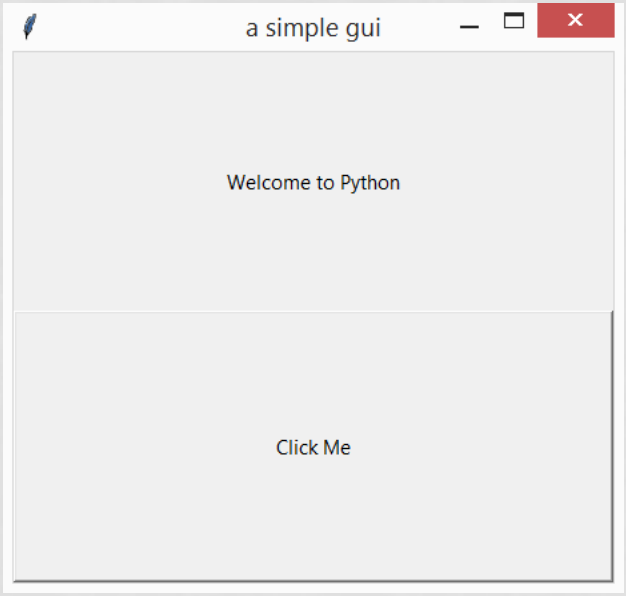
EVENTS

- in Tk there is an event loop which receives events from the OS like mouse and keyboard actions
- event drive programming: often you want a widget to execute a command, for this you must install a handler or callback function
- for events that don't have a command callback associated with them, you can you can bind an event to an anonymous function

GEOMETRY MANAGERS

- pack : using constraints
  - widgets are placed in order of definition
  - o.k. for simple applications
- grid : places widgets into cells of invisible grid
  - based on rows and columns
  - you can place widgets in multiple rows/columns
  - easiest manager ?
- place : allows precise positioning of widgets
  - using absolute positions

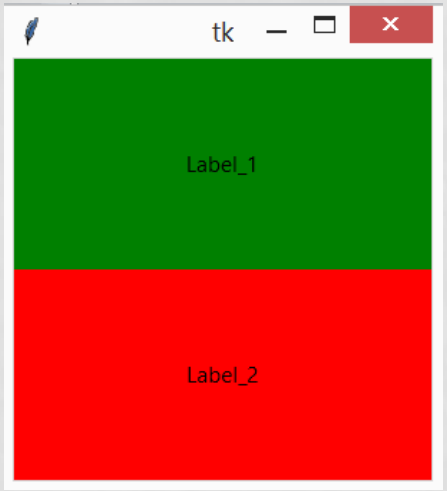
```
1  from tkinter import *
2
3  window = Tk() # create root window
4  window.title('a simple gui')
5  label = Label(window, text = "Welcome to Python")
6  button = Button(window, text = "Click Me")
7  # geometry manager pack : pack vertically, expand with window,
8  # fill both in x & y direction
9  label.pack(side=TOP, expand=YES, fill=BOTH)
10 button.pack(side=TOP, expand=YES, fill=BOTH)
11
12 # wait for events
13 window.mainloop()
```



EXPLANATION

- side : which side of widget's parent to place the widget
  - to pack widgets vertically use TOP
  - to pack widgets horizontally, use LEFT
- expand : if true, widget expands as wide as the parent widget to fill any space not otherwise used in widget's parent
- fill : if true, widget will occupy all space provided by the widget's parent
  - fill horizontally, fill vertically or both directions

```python
from tkinter import *

root = Tk()
root.geometry('200x200+200+200')

label_1 = Label(root, text='Label_1', bg='green')
label_2 = Label(root, text='Label_2', bg='red')

label_1.pack(expand=YES, fill=BOTH)
label_2.pack(expand=YES, fill=BOTH)

root.mainloop()
```

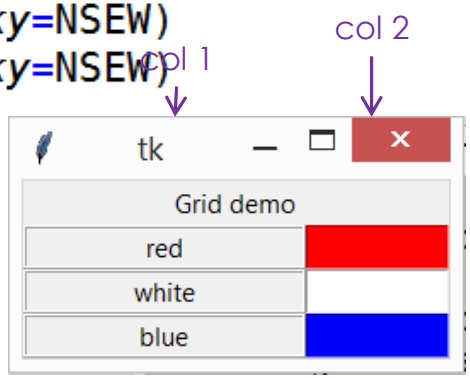



```python
from tkinter import *

root = Tk()
root.geometry('200x200+200+200')

label_1 = Label(root, text='Label_1', bg='green')
label_2 = Label(root, text='Label_2', bg='red')

label_1.pack(expand=YES, fill=Y)
label_2.pack(expand=NO, fill=X)

root.mainloop()
```

ROOT.GEOMETRY

- size : width x height + x_offset + y_offset:
  `root.geometry("170x200+30+30")`

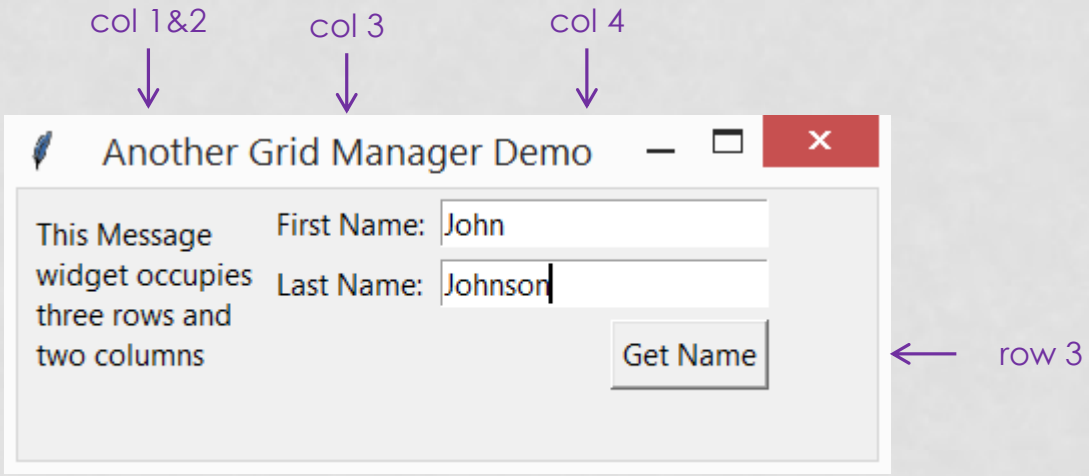
```
1  from tkinter import *
2  colors = ['red', 'white', 'blue']
3
4  root = Tk()
5  lab = Label(root, text='Grid demo')
6  lab.grid(columnspan=2)
7
8  row = 1
9  for color in colors:
10     lab = Label(root, text=color, relief=RIDGE, width=20)
11     ent = Entry(root, bg=color, relief=SUNKEN, width=10)
12     lab.grid(row=row, column=0, sticky=NSEW)
13     ent.grid(row=row, column=1, sticky=NSEW)
14     root.rowconfigure(row, weight=1)
15     row += 1
16
17 root.columnconfigure(0, weight=0)
18 root.columnconfigure(1, weight=1)
19 mainloop()
```



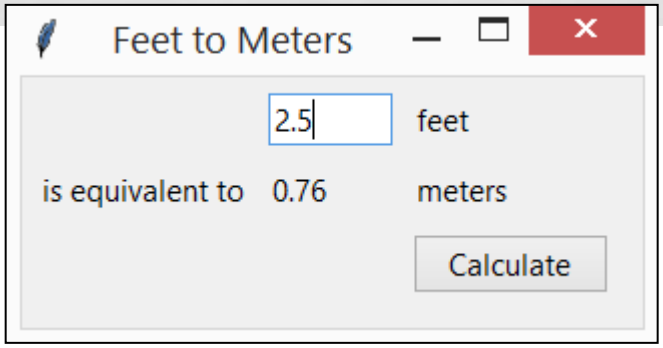
EXPLANATION

- grid cell sizes : based on largest widget in the row or column
- width sizes are in pixels
- Entry() is a text entry field
- relief : 3-D effects around the widget (RAISED, SUNKEN, ...)
- row- and column configure: is resizing allowed, and if yes, how fast
  - weight = 0 means not expandable
  - weight = 3 means : 3 times faster than weight = 1
- sticky : how the widget will expand
  - lets you force an element to stick to one side and leave empty space on the other side
  - widgets can be made sticky in four dimensions : N, S, E, W
  - W means: anchor the widget to the left side of the cell
  - WE anchors it to both the left & right side (stretch horizontally)

```
1  from tkinter import *
2
3  root = Tk()
4  root.title("Another Grid Manager Demo")
5
6  mes = Message(root, text = "This Message widget occupies 3 rows & 2 columns")
7  mes.grid(row = 1, column = 1, rowspan = 3, columnspan = 2)
8
9  # padding is the space between border and contents of a cell
10 Label(root, text = "First Name:").grid(row = 1, column = 3)
11 Entry(root).grid(row = 1, column = 4, padx = 5, pady = 5)
12 Label(root, text = "Last Name:").grid(row = 2, column = 3)
13 Entry(root).grid(row = 2, column = 4)
14 Button(root, text = "Get Name").grid(row = 3, padx = 5, pady = 5, column = 4, sticky = E)
15
16 root.mainloop()
```



```python
from tkinter import *
from tkinter import ttk

def calculate():
    value = float(feet.get())
    meters.set('{:.2f}'.format(0.3048 * value))

root = Tk()
root.title("Feet to Meters")

feet = StringVar()
meters = StringVar()

mainframe = ttk.Frame(root, padding="3 3 12 12") # left top right bottom
mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
mainframe.columnconfigure(0, weight=1)
mainframe.rowconfigure(0, weight=1)

# bind the entry widget to a StringVar object
feet_entry = ttk.Entry(mainframe, width=7, textvariable=feet)
feet_entry.grid(column=2, row=1, sticky=(W, E))
ttk.Label(mainframe, textvariable=meters).grid(column=2, row=2, sticky=(W, E))
# command defines handler function
ttk.Button(mainframe, text="Calculate", command=calculate).grid(column=3, row=3, sticky=W)
ttk.Label(mainframe, text="feet").grid(column=3, row=1, sticky=W)
ttk.Label(mainframe, text="is equivalent to").grid(column=1, row=2, sticky=E)
ttk.Label(mainframe, text="meters").grid(column=3, row=2, sticky=W)
for child in mainframe.winfo_children():
    child.grid_configure(padx=5, pady=5)

feet_entry.focus()
# bind return to handler function
root.bind('<Return>', calculate)
root.mainloop()
```



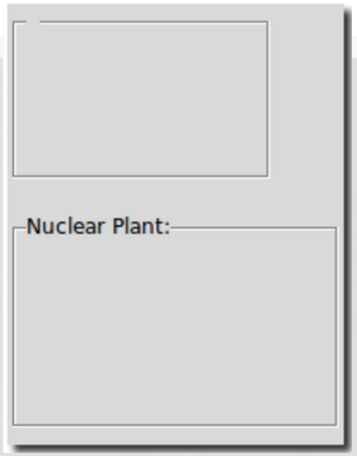
CONTROL VARIABLES

```
v = tk.StringVar()  # Holds a string; default value ''
v = tk.IntVar()     # Holds an int; default value 0
v = tk.DoubleVar()  # Holds a float; default value 0.0
```

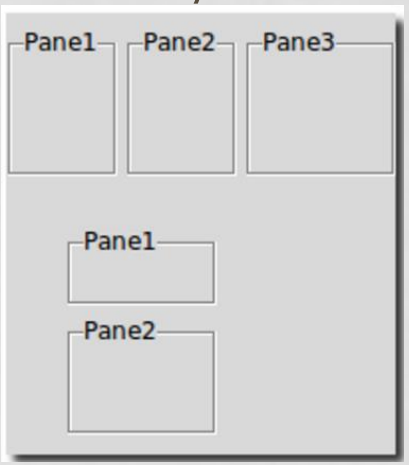
- text displayed in an Entry widget is linked to a control variable
- a single control variable can be shared by a group widgets
- control variables have a get() and set() method
- you have to initialize tkinter before you can use tkinter objects:

```
root = Tk()
feet = StringVar()
```

ORGANIZING COMPLEX INTERFACES



- multiple label frames
  - lf = ttk.Labelframe(*parent*, text='Label')
  - example
- paned windows with label frames
  - stack two or more resizable widgets above or below each other
  - p = ttk.Panedwindow(*parent*, orient=VERTICAL)
  - example



AGENDA

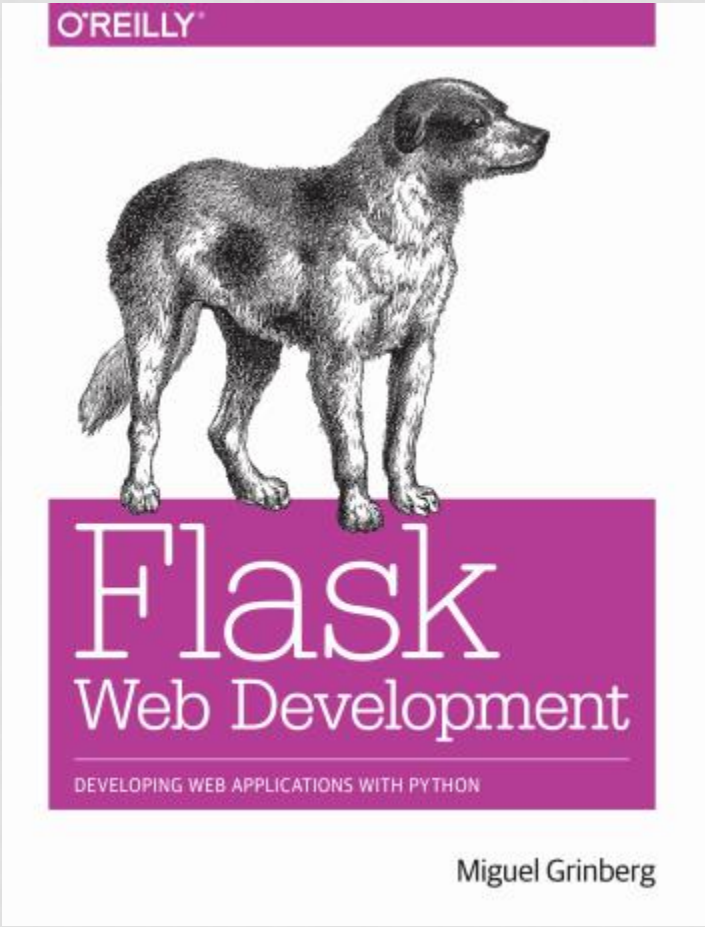
- Python documentation
- case: word histogram
- recursion
- case: solving Numbrix
- PySerial
- tkinter GUI-toolkit
- **web-programming**

HOTFRAMEWORKS.COM

Rankings

Framework	Github Score	Stack Overflow Score	Overall Score
ASP.NET		100	100
AngularJS	100	95	97
Ruby on Rails	95	98	96
ASP.NET MVC		93	93
Django	90	92	91
Meteor	96	78	87
Laravel	92	83	87
Spring	82	90	86
Express	93	79	86
CodeIgniter	86	85	85
Symfony	86	84	85
Ember.js	89	78	83
Flask	90	74	82

FLASK


O'REILLY
Flask
Web Development
DEVELOPING WEB APPLICATIONS WITH PYTHON
Miguel Grinberg