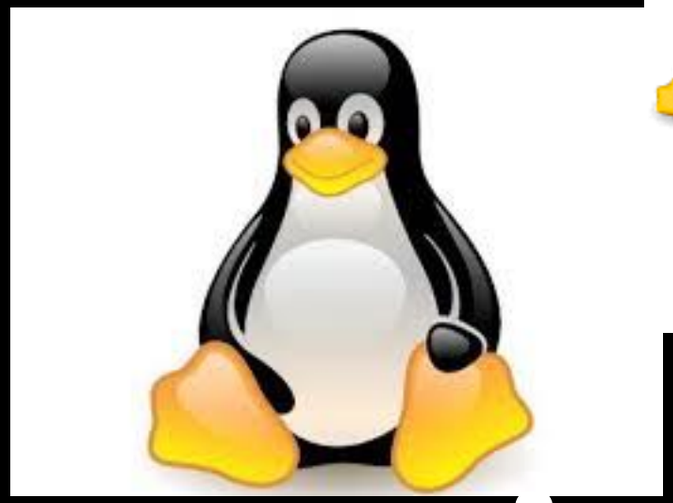
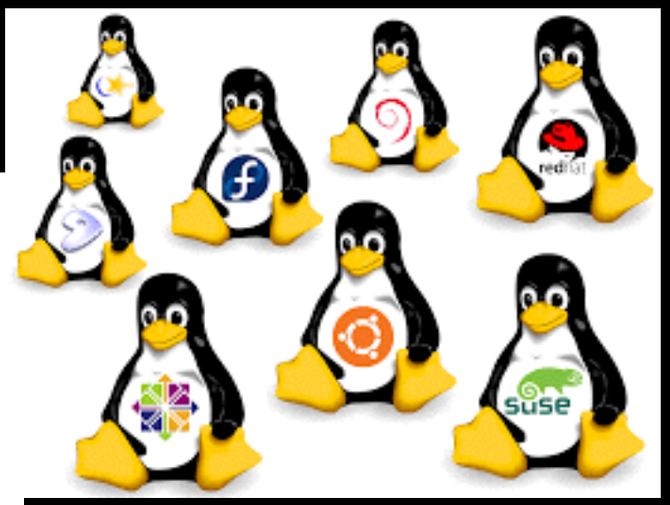
Programming The Hard Way

Workshop TI Unila - 17 Nov 2017

Agenda

- Operating System: **Linux**
- Text Editor: **Emacs**
- Programming Language: **Python & Haskell**





Linux

Secure | https://linux.slashdot.org/story/17/11/14/2223227/all-500-of-the-worlds-top-500-supercomputers-are-running-linux

All 500 of the World's Top 500 Supercomputers Are Running Linux (zdnet.com)

279

Posted by **BeauHD** on Tuesday November 14, 2017 @08:25PM from the it's-about-time dept.

Freshly Exhumed shares a report from ZDnet:

> Linux rules supercomputing. This day has been coming since 1998, when Linux first appeared on the TOP500 Supercomputer list. Today, it finally happened: All 500 of the world's fastest supercomputers are running Linux. The last two non-Linux systems, a pair of Chinese IBM POWER computers running AIX, dropped off the November 2017 TOP500 Supercomputer list. When the first TOP500 supercomputer list was compiled in June 1993, Linux was barely more than a toy. It hadn't even adopted Tux as its mascot yet. It didn't take long for Linux to start its march on supercomputing.
>
> From when it first appeared on the TOP500 in 1998, Linux was on its way to the top. Before Linux took the lead, Unix was supercomputing's top operating system. Since 2003, the TOP500 was on its way to Linux domination. By 2004, Linux had taken the lead for good. This happened for two reasons: First, since most of the world's top supercomputers are research machines built for specialized tasks, each machine is a standalone project with unique characteristics and optimization requirements. To save costs, no one wants to develop a custom operating system for each of these systems. With Linux, however, research teams can easily modify and optimize Linux's open-source code to their one-off designs.

The semiannual TOP500 Supercomputer List was released yesterday. It also shows that China now claims 202 systems within the TOP500, while the United States claims 143 systems.

```
git clone https://github.com/torvalds/linux
cd linux
git log --pretty=oneline
```

```
yum groupinstall "Development Tools"
yum install ncurses-devel
yum install qt-devel
yum install unifdef
```

```
cd /usr/src/kernels/*
make modules install
make install
```

www.buildyourownlinux.com

Build Your Own Linux

"Build Your Own Linux (From Scratch)" walks users through building a basic Linux distribution. Presented by Linux Academy & Cloud Assessments. Access the main Linux Academy website to view related course videos and other content, and the Cloud Assessments website for free cloud training powered by AI.

Join the Linux Academy community for free to chat with thousands of like-minded Linux experts.

Section 1

Our Goal

WHAT WE ARE BUILDING

This course walks through the creation of a 64-bit system based on the Linux kernel. Our goal is to produce a small, sleek system well-suited for hosting containers or being employed as a virtual machine.

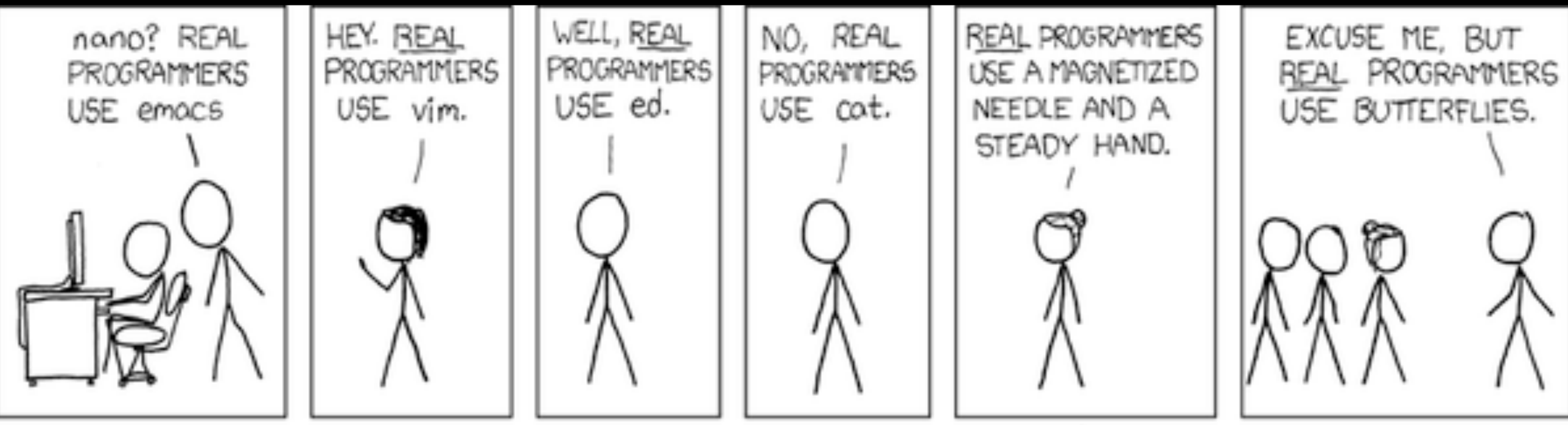
Because we don't need every piece of functionality under the sun, we're not going to include every piece of software you might find in a typical distro. This distribution is intended to be minimal.

Here is what our end-result will look like:

- 64-bit Linux 4.8 Kernel with GCC 6.2 and glibc 2.24
- A system compatible with both EFI and BIOS hardware

Demo: Linux Install with VirtualBox

The highly extensible text editor: **Emacs**


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
"
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

emacs.sexy

Emacs is sexy!

Description Why Emacs? Installing Emacs Learn Emacs Resources Be free

W acs?

Emacs is a very powerful text prccessor, giving you the power to manipulate documents quickly and efficiently. You can easily move through and edit paragraphs, sentences, words, and logical blocks; blaze through text using powerful search tools; and easily edit thousands of lines at once using regular expressions, keyboard macros and more.

Colorful text editor

Emacs can be customized in every conceivable way, including its looks. You can strip it down, choose between dozens of easy to install themes with `M-x load-theme`, or even create your own and share it with your friends. Here are a couple of nice theme galleries: Emacs Themes, Emacs Theme Gallery.

Et tu, Programmer?

There are tools for every programming language out there. Lisp, Ruby, Python, PHP, Java, Erlang, JavaScript, C, C++, Prolog, Tcl, AWK, PostScript, Clojure, Scala, Perl, Haskell, Elixir all of these languages and more are supported in Emacs. Because of the powerful Lisp core, Emacs is easy to extend to add support for new languages if the urge strikes you.

You get lots of features out of the box, including syntax highlighting, automatic indentation, REPL support, debugging, code browsing, version control integration and much more.

More!

`Org mode` helps you to keep notes, maintain TODO lists, plan projects and author documents. You can use your Org documents to create HTML websites like this one or export to LaTeX, Beamer, OpenDocuments and many other formats.

`Tramp` allows you to edit remote files without leaving Emacs. You can seamlessly edit files on remote servers via SSH or FTP, edit local files with su/sudo, and much more.

`M-x butterfly` unleashes the powers of the butterfly. The real way of programming.

Use the built in IRC client `ERC` along with `BitlBee` to connect to your favorite chat services, or use the Jabber package to hop on any XMPP service.

Out of the box Emacs includes a mail client, web browser, calendar, and games; you can even edit video and images inside Emacs. There are more than 2,000 packages for Emacs, and more are written all the time. You can easily extend your Emacs with new packages from GNU ELPA, MELPA and Marmelade repositories.

HOW TO LEARN EMACS

A beginner's guide to Emacs 24 or later · http://j.mp/beginemacs
Sacha Chua (@sachac) LivingAnAwesomeLife.com

Questions? I'd love to hear from you!

May 17 2013

If you're a developer or sysad...

Learn Vim

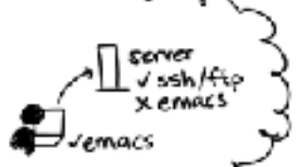
the other text editor

Seriously. Learn the basics so that you can easily work on other people's computers. If you know your way around Vim, people won't give you as much grief over Emacs.

Here's what you need to know:

i	insert mode ↔ <Esc> command mode
	:vimtutor
★Are you a long-time vi user trying out Emacs? Check out Evil (search for "Evil Emacs")!	:w write/save file
	:q quit
	:q! really quit

★ You can actually edit remote files in Emacs without installing Emacs on the other computer, but that's an intermediate topic. (see TRAMP)



Okay. Once you know the basics of vim, you can get on with learning Emacs. :)

Why learn Emacs?
→ customizable
→ endless room for growth

Why are Emacs terms so weird? It's because Emacs has been around for a very long time, and it's hard to change the way things are called. Don't worry, you'll get used to it.

Learn the terms

This will help you read documentation.

Reading keyboard shortcuts:

C-x C-s → press Ctrl+x, then Ctrl+s

C (Control Key) - x C (Control Key) - s

(you can keep Ctrl pressed, then press X and then S)

I use a mix of beginner-friendly and Emacs-standard terms in this guide.

You'll also see keyboard shortcuts like:

M-x → Press Alt+x (or ⌘x)
Meta Key (Alt / ⌘)

★ This lets you execute commands by name. (even better with M-x icomplete-mode)

★ Alternatively, you can replace M-... with <Esc> ... (ex: press Esc, then press x)

RET → Return / Enter Key

★ Press <Tab> at a prompt to see options.

Windows show buffers, which could be:
- a file
- a process
- info not associated with a file

Learn how to learn more

Inside Emacs:

C-h t Tutorial
C-h i Info manual (check out the menus) (works with menus too)
C-h K <keyboard shortcut> Describes a shortcut
C-h a Searches commands
C-h w <command> Shows shortcut
C-h m Describe current modes
C-h C-h Help on help
C-h f Describe a function (by name)

On the web:
EmacsWiki.org
planet.emacsen.org
stackoverflow.com
reddit.com/r/emacs
& more!

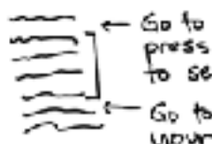
Like IRC? Check out #emacs on irc.freenode.net
<3 <3 <3 folks are wonderful and helped me put this together!

Learn Emacs basics

C-x C-f open/visit a file
C-x C-s save/write a file
C-x C-c quit

★ Note: you don't need to quit Emacs after each file. Just use C-x C-f to open the next one.

How to select text:



← Go to the start and press C-SPC (Ctrl+space) to set the mark (the beginning)

← Go to the end and run your command. (Ex: C-w is "kill", which actually cuts the text.)

Text editing

C-/ undo (need to redo? just do something unrelated, then undo the undo) ★ even better with undo-tree
C-w kill/cut
C-y yank/paste → learn about the kill ring
M-x replace-string RET

★ Learn how to use keyboard macros. They're awesome.

C-x (start macro
C-x) end macro
C-x e execute macro
e ... ...again

...and more!

Buffer & window management

C-x b switch buffer → even better with M-x ido-mode
C-x 2 split
C-x 3 split
C-x o other window
C-x 0 get rid of current window
C-x 1 get rid of other windows

Navigation & search

M-g M-g go to line (Hold Alt/⌘ and then press g twice)
C-s Interactive search
C-r Interactive search backward
M-> End of buffer
M-< Beginning of buffer
M-x occur RET Find lines

Extend & customize

M-x load-theme RET
Try out color themes → use Elpa to see the list, and list-packages to add more
M-x customize-group RET
set common options
M-x customize-face RET
change background, foreground, etc.

M-x list-packages RET
install lots of modules...
→ Want a prettier Emacs? Check out color themes. (Lots of people like color-theme-solarized)

and then...

editing your ~/.emacs file!

initializes your Emacs, adds new functionality, and so on.
} See emacswiki.org for lots of examples

Use M-x eval-buffer or restart Emacs to see the changes.

Broke your Emacs config? } emacs -q skips ~/.emacs

Other good things to learn

Org-mode.org
organize your life in plain text

Narrowing / Widening (focus)

Calc
powerful calculator and converter

Eshell / Term
command-line in Emacs

TRAMP
remote access

Writing & debugging Emacs Lisp (edebug)
(it sounds scary, but it's powerful :D)

There's so much more!

Ask away, and discover more by exploring!

Sacha Chua

Python & Haskell

Python Cheat Sheet

JUST THE BASICS

Created By: Arianne Colton and Sean Chen

General

- Python is case sensitive
- Python index starts from 0
- Python uses whitespace (tabs or spaces) to indent code instead of using braces.

HELP

Help Home Page	`help()`
Function Help	`help(str.replace)`
Module Help	`help(re)`

MODULE (AKA LIBRARY)

Python module is simply a '.py' file

List Module Contents	`dir(module1)`
Load Module	`import module1` *
Call Function from Module	`module1.func1()`

* 'import' statement creates a new namespace and executes all the statements in the associated '.py' file within that namespace. If you want to load the module's content into current namespace, use 'from module1 import *'

Scalar Types

Check data type : `type(variable)`

SIX COMMONLY USED DATA TYPES

1. **int/long*** - Large int automatically converts to long
2. **float*** - 64 bits, there is no 'double' type
3. **bool*** - True or False
4. **str*** - ASCII valued in Python 2.x and Unicode in Python 3
   - String can be in single/double/triple quotes
   - String is a sequence of characters, thus can be treated like other sequences
   - Special character can be done via \ or preface with r

```
str1 = r'this\test'
```

   - String formatting can be done in a number of ways

```
template = '%.2f %s haha $%d';
str1 = template % (4.88, 'hola', 2)
```

Scalar Types

* str(), bool(), int() and float() are also explicit type cast functions.

5. **NoneType(None)** - Python 'null' value (ONLY one instance of None object exists)
   - None is not a reserved keyword but rather a unique instance of 'NoneType'
   - None is common default value for optional function arguments :

```
def func1(a, b, c = None)
```

   - Common usage of None :

```
if variable is None :
```

6. **datetime** - built-in python 'datetime' module provides 'datetime', 'date', 'time' types.
   - 'datetime' combines information stored in 'date' and 'time'

Create datetime from String	`dt1 = datetime.strptime('20091031', '%Y%m%d')`
Get 'date' object	`dt1.date()`
Get 'time' object	`dt1.time()`
Format datetime to String	`dt1.strftime('%m/%d/%Y %H:%M')`
Change Field Value	`dt2 = dt1.replace(minute=0, second=30)`
Get Difference	`diff = dt1 - dt2` # diff is a 'datetime.timedelta' object

Note : Most objects in Python are mutable except for 'strings' and 'tuples'

Data Structures

Note : All non-Get function call i.e. `list1.sort()` examples below are in-place (without creating a new object) operations unless noted otherwise.

TUPLE

One dimensional, fixed-length, **immutable** sequence of Python objects of ANY type.

Create Tuple	`tup1 = 4, 5, 6` or `tup1 = (6,7,8)`
Create Nested Tuple	`tup1 = (4,5,6), (7,8)`
Convert Sequence or Iterator to Tuple	`tuple([1, 0, 2])`
Concatenate Tuples	`tup1 + tup2`
Unpack Tuple	`a, b, c = tup1`

Application of Tuple

Swap variables	`b, a = a, b`

Data Structures

LIST

One dimensional, variable length, **mutable** (i.e. contents can be modified) sequence of Python objects of ANY type.

Create List	`list1 = [1, 'a', 4]` or `list1 = list(tup1)`
Concatenate Lists*	`list1 + list2` or `list1.extend(list2)`
Append to End of List	`list1.append('b')`
Insert to Specific Position	`list1.insert(posIdx, 'b')` **
Inverse of Insert	`valueAtIdx = list1.pop(posIdx)`
Remove First Value from List	`list1.remove('a')`
Check Membership	`3 in list1` -> True ***
Sort List	`list1.sort()`
Sort with User-Supplied Function	`list1.sort(key = len)` # sort by length

* List concatenation using '+' is expensive since a new list must be created and objects copied over. Thus, `extend()` is preferable.

** Insert is computationally expensive compared with append.

*** Checking that a list contains a value is lot slower than dicts and sets as Python makes a linear scan where others (based on hash tables) in constant time.

Built-in 'bisect' module‡

- Implements binary search and insertion into a sorted list
- 'bisect.bisect' finds the location, where 'bisect.insort' actually inserts into that location.

‡ WARNING : bisect module functions do not check whether the list is sorted, doing so would be computationally expensive. Thus, using them in an unsorted list will succeed without error but may lead to incorrect results.

SLICING FOR SEQUENCE TYPES†

† Sequence types include 'str', 'array', 'tuple', 'list', etc.

Notation	`list1[start:stop]`
	`list1[start:stop:step]` (If step is used ‡)

Note :
- 'start' index is included, but 'stop' index is NOT.
- start/stop can be omitted in which they default to the start/end.

* Application of 'step' :

Take every other element	`list1[::2]`
Reverse a string	`str1[::-1]`

DICT (HASH MAP)

Create Dict	`dict1 = {'key1' :'value1', 2 :[3, 2]}`
Create Dict from Sequence	`dict(zip(keyList, valueList))`
Get/Set/Insert Element	`dict1['key1']` `dict1['key1'] = 'newValue'`
Get with Default Value	`dict1.get('key1', defaultValue)` **
Check If Key Exists	`'key1' in dict1`
Delete Element	`del dict1['key1']`
Get Key List	`dict1.keys()` ***
Get Value List	`dict1.values()` ***
Update Values	`dict1.update(dict2)` # dict1 values are replaced by dict2

* 'KeyError' exception if the key does not exist.

** 'get()' by default (aka no 'defaultValue') will return 'None' if the key does not exist.

*** Returns the lists of keys and values in the same order. However, the order is not any particular order, aka it is most likely not sorted.

Valid dict key types

- Keys have to be immutable like scalar types (int, float, string) or tuples (all the objects in the tuple need to be immutable too)
- The technical term here is 'hashability', check whether an object is hashable with the `hash('this is string')`, `hash([1, 2])` - this would fail.

SET

- A set is an **unordered** collection of UNIQUE elements.
- You can think of them like dicts but keys only.

Create Set	`set([3, 6, 3])` or `{3, 6, 3}`
Test Subset	`set1.issubset(set2)`
Test: Superset	`set1.issuperset(set2)`
Test sets have same content	`set1 == set2`

- **Set operations :**

Union (aka 'or')	`set1 | set2`
Intersection (aka 'and')	`set1 & set2`
Difference	`set1 - set2`
Symmetric Difference (aka 'xor')	`set1 ^ set2`

Functions

Python is **pass by reference**, function arguments are passed by reference.

- Basic Form :

```
def func (posArg , keywordArg =
), ..):
```

Note :
- Keyword arguments MUST follow positional arguments.
- Python by default is NOT "lazy evaluation", expressions are evaluated immediately.

- Function Call Mechanism :
  1. All functions are local to the module level scope. See 'Module' section.
  2. Internally, arguments are packed into a tuple and dict, function receives a tuple 'args' and dict 'kwargs' and internally unpack.
- Common usage of 'Functions are objects' :

```
def func (ops = _str.strip, user_
define_func, ..], ..):
    for function in ops:
        value = function(value)
```

RETURN VALUES

- **None** is returned if end of function is reached without encountering a return statement.
- Multiple values return via ONE tuple object

```
return (value1, value2)
value1, value2 = func(..)
```

ANONYMOUS (AKA LAMBDA) FUNCTIONS

- What is Anonymous function?
  A simple function consisting of a single statement.

```
lambda x : x * 2
# def func(x) : return x * 2
```

- Application of lambda functions : 'currying' aka deriving new functions from existing ones by partial argument application.

```
ma60 = lambda x : pd.rolling_mean(x,
60)
```

USEFUL FUNCTIONS (FOR DATA STRUCTURES)

1. **Enumerate** returns a sequence (i, value) tuples where i is the index of current item.

```
for i, value in enumerate(collection):
```

   - Application : Create a dict mapping of value of a sequence (assumed to be unique) to their locations in the sequence.
2. **Sorted** returns a new sorted list from any sequence

```
sorted([2, 1, 3,] -> [1, 2, 3]
```

   - Application :

```
sorted(set('abc bcd')) =>  ' ',
'a', 'b', 'c', 'd']
# returns sorted unique characters
```

3. **Zip** pairs up elements of a number of lists, tuples or other sequences to create a list of tuples :

```
zip(seq1, seq2) =>
[('seq1 1', 'seq2 1'), (..), ..]
```

   - Zip can take arbitrary number of sequences. However, the number of elements it produces is determined by the 'shortest' sequence.
   - Application : Simultaneously iterating over multiple sequences :

```
for i, (a, b) in
enumerate(zip(seq1, seq2)):
```

   - Unzip - another way to think about this is converting a list of rows to a list of columns.

```
seq1, seq2 = zip(*zipOutput)
```

4. **Reversed** iterates over the elements of a sequence in reverse order.

```
list(reversed(range(10)))
```

* reversed() returns the iterator, list() makes it a list.

Control and Flow

1. Operators for conditions in 'if/else' :

Check if two variables are same object	`var1 is var2`
... are different object	`var1 is not var2`
Check if two variables have same value	`var1 == var2`

WARNING : Use 'and', 'or', 'not' operators for compound conditions, not &&, ||, !.

2. Common usage of 'for' operator :

Iterating over a collection (i.e. list or tuple), or an iterator	`for element in iterator :`
... If elements are sequences, can be 'unpack'	`for a, b, c in iterator :`

3. 'pass' - no-op statement. Used in blocks where no action is to be taken.
4. Ternary Expression - aka less verbose 'if/else'
   - Basic Form :

```
value = true-expr if condition
else false-expr
```

5. No switch/case statement, use if/elif instead

Object-Oriented Programming

1. **'object'** is the root of all Python types
2. Everything (number, string, function, class, module, etc.) is an object, each object has a 'type'. Object variable is a pointer to its location in memory.
3. All objects are reference-counted.

```
sys.getrefcount(5)  -> x
a = 5; b = 5
# This creates a 'reference' to the object on the
right side of =, thus both a and b point to 5
sys.getrefcount(5)  -> x + 2
del(a); sys.getrefcount(5) => x - 1
```

4. **Class** Basic Form :

```
class MyObject(object):
    # 'self' is equivalent of 'this' in Java/C++
    def __init__(self, name):
        self.name = name
    def memberFunc1(self, arg1):
        ..
    @staticmethod
    def classFunc2(arg1):
        ..
obj1 = MyObject('name1')
obj1.memberFunc1('a')
MyObject.classFunc2('b')
```

5. Useful interactive tool :

```
dir(variable1)  # list all methods available on
the object
```

Common String Operations

Concatenate List/Tuple with Separator	`', '.join(['v1', 'v2', 'v3']) => 'v1, v2, v3'`
Format String	`string1 = 'My name is {0} {name}'` `newString1 = string1.format('Sean', name = 'Chen')`
Split String	`sep = '-';` `stringList1 = string1.split(sep)`
Get Substring	`start = 1;  string1[start:5]`

String Padding with Zeros	`month = '5';`
	`month.zfill(2) => '05'`
	`month = '12';`
	`month.zfill(2) -> '12'`

Exception Handling

1. Basic Form :

```
try:
    ..
except ValueError as e:
    print e
except (TypeError, AnotherError):
    ..
except:
    ..
finally:
    .. # clean up, e.g. close db
```

2. Raise Exception Manually

```
raise AssertionError # assertion failed
raise SystemExit     # request program exit
raise RuntimeError('Error message :
..')
```

List, Set and Dict Comprehansions

Syntactic sugar that makes code easier to read and write

1. **List comprehensions**
   - Concisely form a new list by filtering the elements of a collection and transforming the elements passing the filter in one concise expression.
   - Basic form :

```
[expr for val in collection if condition]
```

   A shortcut for :

```
result = []
for val in collection:
    if condition:
        result.append(expr)
```

   The filter condition can be omitted, leaving only the expression.
2. **Dict Comprehension**
   - Basic form :

```
{key-expr : value-expr for value in
collection if condition}
```

3. **Set Comprehension**
   - Basic form : same as List Comprehension except with curly braces instead of [].
4. **Nested list Comprehensions**
   - Basic form :

```
[expr for val in collection for
innerVal in val if condition]
```

Created by Arianne Colton and Sean Chen
data.scientist.info@gmail.com

Based on content from
'Python for Data Analysis' by Wes McKinney

Updated: May 3, 2016

Imperative (Procedureal) vs Functional

Imperative/OOP:
- Encapsulation
- Inheritance
- Polymorphism

Functional Programming:
- Lambda Calculus
- Higher Order Function
- Immutability
- No side-effects

Demo: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html