PythonFu

Arnaud Loonstra, 12-08-2016, Amsterdam

Preparation

- Install Python 3 (latest version, comes preinstalled on most Linux distros)
- Install virtualenv
- Make sure you are connected to the Internet
- All used data can be found at http://github.com/sphaero/workshops

Command Line Interface

ls	list dir content
cd	change current dir
cp	copy
rm	remove/delete
nano	text editor
vi	text editor (difficult)
dmesg	output kernel messages
tail	last contents of a file
more	file pager
less	file pager (nicer)
cat	binary output a file
man	manuals
mkdir	create a directory
touch	update/create a file (access time)
pwd	present working dir
ln	create (symbolic) link
du	disk use
chmod	change permission
fg/bg	fore-/background a process

CONTROL:

CTRL-C	Kill
CTRL-D	Exit
CTRL-Z	suspend
CTRL-S	stop input (undo CTRL-Q)

TAB	command completion!!!!

Commands can be put in a file for batching:

```
#!/bin/bash
pwd
cd /tmp
touch hoi
echo hallo > hoi
tail hoi
cd -
```

Display characterization

```
print('hello world')
```

Source code example

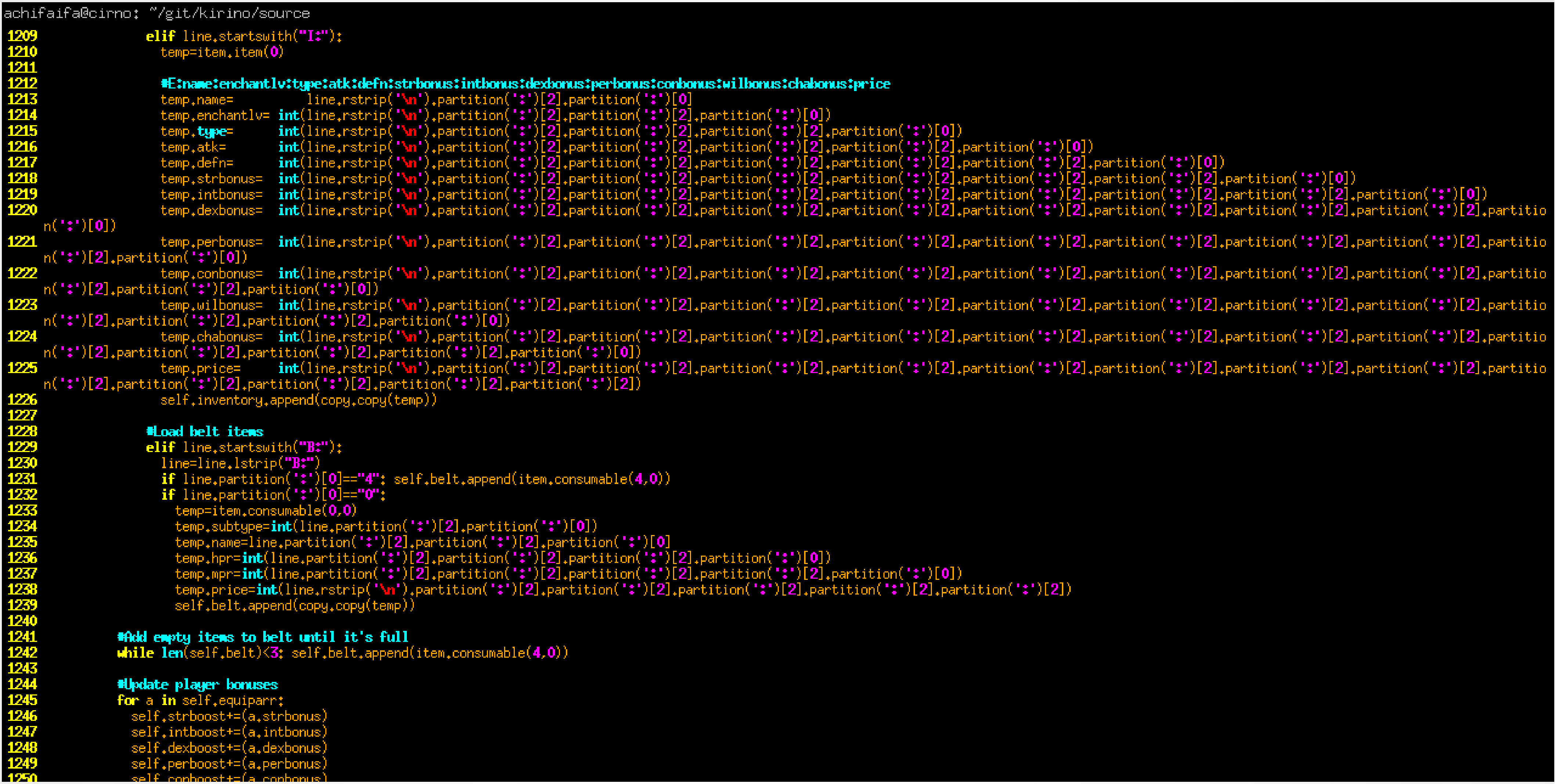
```
$ export PYTHONHOME=“/tmp“
$ python run.py
```

Shell commands example (terminal)

```
>>> def log(msg):
...     print(msg)
...
>>> log("hello world")
hello world
>>>
```

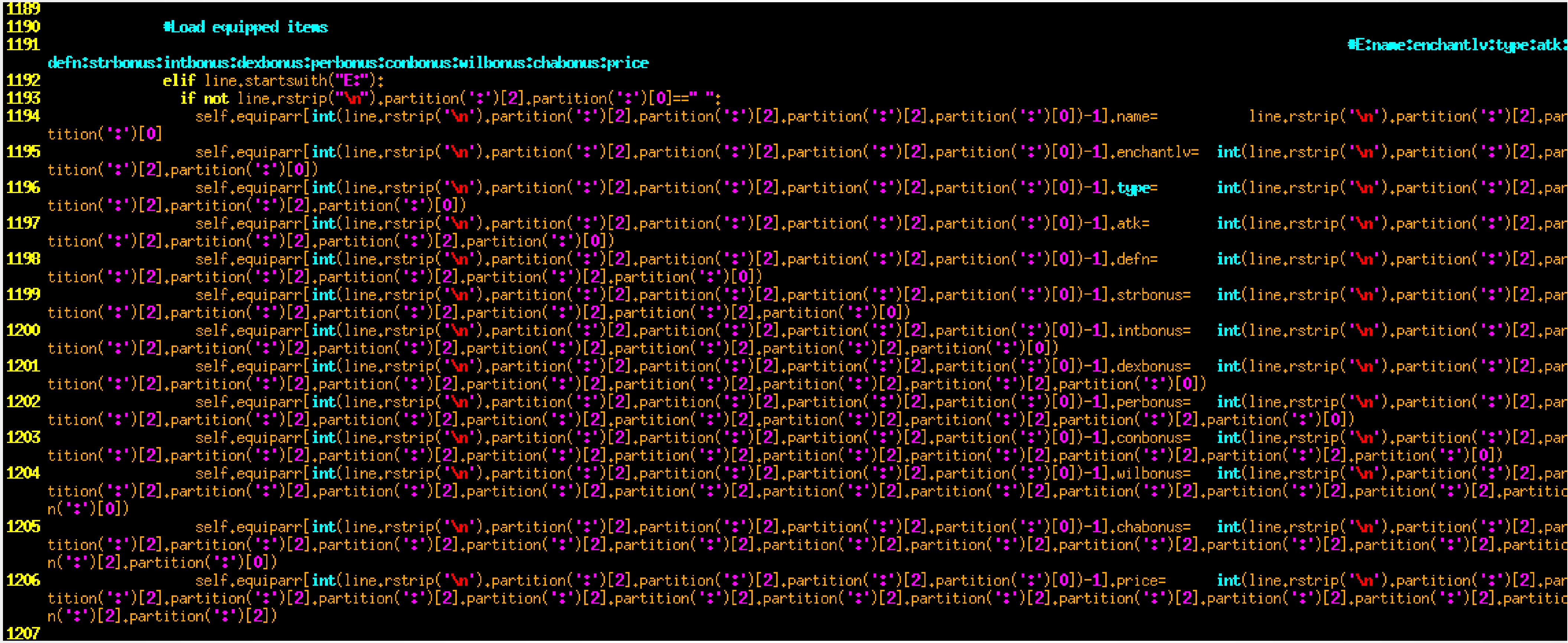
Python interpreter example

???



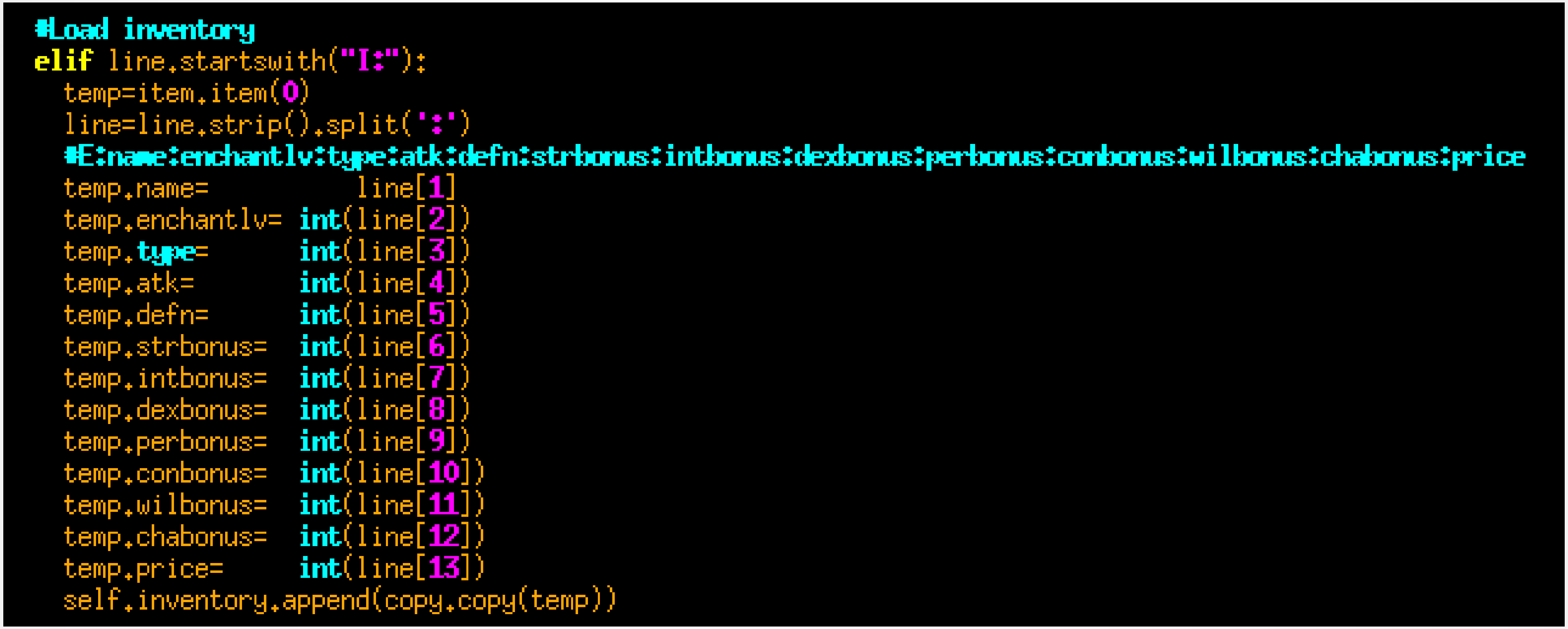
```
achifaifa@cirno: ~/git/kirino/source
              elif line.startswith("I:"):
                temp=item.item(0)

                #E:name:enchantlv:type:atk:defn:strbonus:intbonus:dexbonus:perbonus:conbonus:wilbonus:chabonus:price
                temp.name=          line.rstrip('\n').partition(':')[2].partition(':')[0]
                temp.enchantlv= int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[0])
                temp.type=      int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.atk=       int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.defn=      int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.strbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.intbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.dexbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.perbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.conbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.wilbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.chabonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.price=     int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2])
                self.inventory.append(copy.copy(temp))

              #Load belt items
              elif line.startswith("B:"):
                line=line.lstrip("B:")
                if line.partition(':')[0]=="4": self.belt.append(item.consumable(4,0))
                if line.partition(':')[0]=="0":
                  temp=item.consumable(0,0)
                  temp.subtype=int(line.partition(':')[2].partition(':')[0])
                  temp.name=line.partition(':')[2].partition(':')[2].partition(':')[0]
                  temp.hpr=int(line.partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  temp.mpr=int(line.partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  temp.price=int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2])
                  self.belt.append(copy.copy(temp))

          #Add empty items to belt until it's full
          while len(self.belt)<3: self.belt.append(item.consumable(4,0))

          #Update player bonuses
          for a in self.equiparr:
            self.strboost+=(a.strbonus)
            self.intboost+=(a.intbonus)
            self.dexboost+=(a.dexbonus)
            self.perboost+=(a.perbonus)
            self.conboost+=(a.conbonus)
```

???



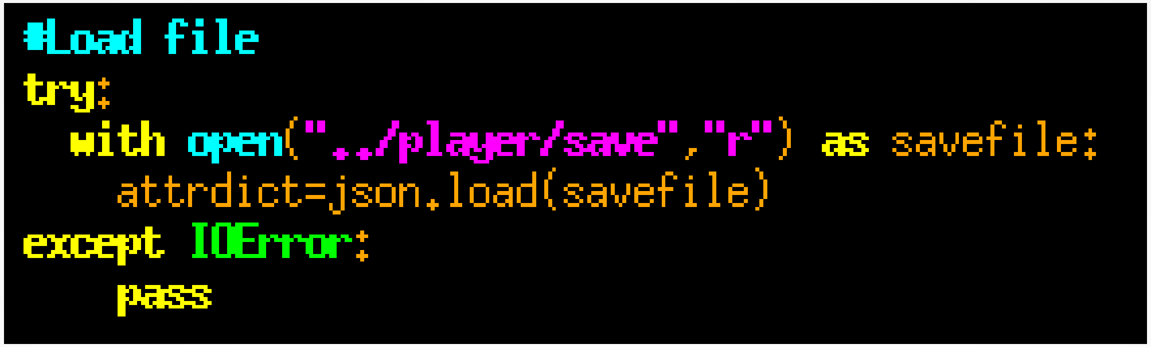
```
        #Load equipped items
                                                                                   #E:name:enchantlv:type:atk:
defn:strbonus:intbonus:dexbonus:perbonus:conbonus:wilbonus:chabonus:price
        elif line.startswith("E:"):
         if not line.rstrip("\n").partition(':')[2].partition(':')[0]==" ":
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].name=      line.rstrip('\n').partition(':')[2].par
tition(':')[0]
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].enchantlv=  int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].type=       int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].atk=        int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].defn=       int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].strbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].intbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].dexbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].perbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].conbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].wilbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].chabonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[2].partition(':')[0])
          self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].price=      int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[2].partition(':')[2])
```

???



```
#Load inventory
elif line.startswith("I:"):
  temp=item.item(0)
  line=line.strip().split(':')
  #E:name:enchantlv:type:atk:defn:strbonus:intbonus:dexbonus:perbonus:conbonus:wilbonus:chabonus:price
  temp.name=          line[1]
  temp.enchantlv= int(line[2])
  temp.type=      int(line[3])
  temp.atk=       int(line[4])
  temp.defn=      int(line[5])
  temp.strbonus=  int(line[6])
  temp.intbonus=  int(line[7])
  temp.dexbonus=  int(line[8])
  temp.perbonus=  int(line[9])
  temp.conbonus=  int(line[10])
  temp.wilbonus=  int(line[11])
  temp.chabonus=  int(line[12])
  temp.price=     int(line[13])
  self.inventory.append(copy.copy(temp))
```

Why did this happen?



```
#Load file
try:
  with open("../player/save","r") as savefile:
    attrdict=json.load(savefile)
except IOError:
    pass
```

- The documentation lists the functions in alphabetical order
- partition() is listed before split()
- I stopped reading as soon as I found the first one

“Yuri Numerov @FOSDEM16”

https://archive.fosdem.org/2016/schedule/event/python_mistakes/

Outline

- Running and using Python
- Object Oriented Programming in Python
- Pythonic Thinking

Python pros & cons

- Easy to code
- Code indentation
- Embeddable
- Batteries included
- Huge community
- Simple binding to C
- Opensource

- Code indentation
- Slow
- Global Interpreter Lock
- Python 2 to 3 problem
- Dynamic types

Python version

On unix variants:

```
$ python --version
```

On unix variants with 2 and 3 installed

```
$ python3 --version
```

On windows:

```
$ python.exe --version
```

Python version

Within Python:

```
>>> import sys
>>> sys.version
'3.4.3+ (default, Oct 14 2015, 16:03:50) \n[GCC 5.2.1
20151010]'
>>> sys.version_info
sys.version_info(major=3, minor=4, micro=3,
releaselevel='final', serial=0)
>>> sys.version_info.major
3
>>> sys.version_info.minor
4
```

Python directory structure

./include	headers for C compilers
./lib/python-wheels	wheels will replace eggs
./lib/python3.4	this is where your modules reside
./lib/python3.4/__pycache__	bytecode versions of python code (ignore)
./lib/python3.4/json	the included json module (example)
./lib/python3.4/site-packages	extra installed non native modules
./bin	the python executables

Python module lookup (path)

From within Python

```
>>> import sys
>>> sys.path
['', '/usr/local/lib/python3.4/dist-packages/python_axolotl-
0.1.35-py3.4.egg', '/usr/local/lib/python3.4/dist-
packages/protobuf-3.0.0b2.post2-py3.4.egg',
'/usr/local/lib/python3.4/dist-packages/pycrypto-2.6.1-py3.4-
linux-x86_64.egg', '/usr/lib/python3/dist-packages',
'/usr/lib/python3.4', '/usr/lib/python3.4/plat-x86_64-linux-
gnu', '/usr/lib/python3.4/lib-dynload',
'/home/aloonstra/.local/lib/python3.4/site-packages',
'/usr/local/lib/python3.4/dist-packages']
```

```
>>> print("\n".join(sys.path))
```

Python module lookup (path)

```
>>> print("\n".join(sys.path))

/usr/local/lib/python3.4/dist-packages/python_axolotl-0.1.35-py3.4.egg
/usr/local/lib/python3.4/dist-packages/protobuf-3.0.0b2.post2-py3.4.egg
/usr/local/lib/python3.4/dist-packages/pycrypto-2.6.1-py3.4-linux-x86_64.egg
/usr/lib/python3/dist-packages
/usr/lib/python3.4
/usr/lib/python3.4/plat-x86_64-linux-gnu
/usr/lib/python3.4/lib-dynload
/home/aloonstra/.local/lib/python3.4/site-packages
/usr/local/lib/python3.4/dist-packages
```

Python module lookup (path)

Adding a module search directory within Python:

```
>>> import sys
>>> sys.path.append('/tmp/my_path')
```

Adding a module search directory from the terminal:

```
$ export PYTHONPATH=”/tmp/my_path”
$ python
```

Or a oneliner:

```
$ PYTHONPATH=”/tmp/my_path” python
```

In case PYTHONPATH was already set:

```
$ PYTHONPATH=”$PYTHONPATH:/tmp/my_path” python
```

Python home

Location of Python files (prefix or home)

```
>>> import sys
>>> sys.prefix
'/usr'
```

Setting the Python prefix (home) from the terminal:

```
$ export PYTHONHOME=”/usr”
$ python
>>> import sys
>>> sys.prefix
'/usr'
```

Or a oneliner:

```
$ PYTHONHOME=”/usr” python -c "import sys; print(sys.prefix)"
```

Virtualenv

see: $ virtualenv --help

```
$ virtualenv testing
Running virtualenv with interpreter /usr/bin/python2
New python executable in testing/bin/python2
Also creating executable in testing/bin/python
Installing setuptools, pip...done.
```

```
$ virtualenv -p python3 testing_py3
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in testing_py3/bin/python3
Also creating executable in testing_py3/bin/python
Installing setuptools, pip...done.
```

```
--system-site-packages
          Give the virtual environment access to the global
          site-packages.
```

Virtualenv

activate

```
$ source testing/bin/activate
[testing] $ python
>>> import sys
>>> sys.prefix
'/home/aloonstra/Documents/projects/workshops/AdvancedPython
/testing'
```

deactivate (in the same terminal)

```
$ [testing] $ deactivate
$
```

shebang

Under {Unix}, if the first two bytes of an {executable} file are "**#!**", the {kernel} treats the file as a script rather than a {machine code} program. The word following the "!" (i.e., everything up to the first {whitespace}) is used as the {pathname} of the {interpreter}.

```
#!/usr/bin/python

import ...
```

better

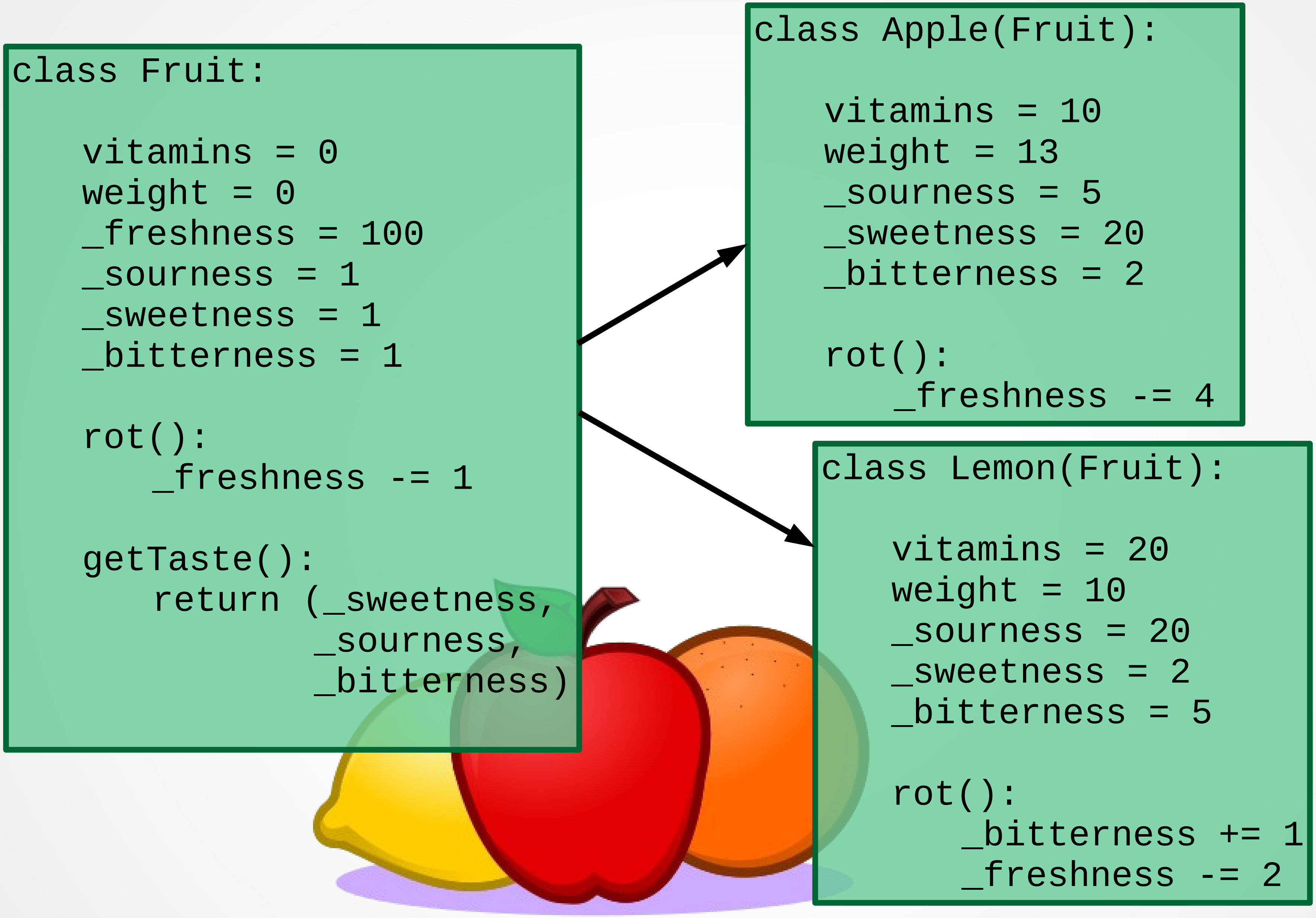
```
#!/usr/bin/env python

import ...
```

Let's code

Outline

- Object Oriented Programming
- Classes & Inheritance
- Tips & Tricks

Classes in Python



```
class Fruit:

    vitamins = 0
    weight = 0
    _freshness = 100
    _sourness = 1
    _sweetness = 1
    _bitterness = 1

    rot():
        _freshness -= 1

    getTaste():
        return (_sweetness,
                _sourness,
                _bitterness)
```

```
class Apple(Fruit):

    vitamins = 10
    weight = 13
    _sourness = 5
    _sweetness = 20
    _bitterness = 2

    rot():
        _freshness -= 4
```

```
class Lemon(Fruit):

    vitamins = 20
    weight = 10
    _sourness = 20
    _sweetness = 2
    _bitterness = 5

    rot():
        _bitterness += 1
        _freshness -= 2
```

Fruit Class

```
class Fruit(object):

    def __init__( self, vit, weight):
        self.vitamins = vit
        self.weight = weight
        self._freshness = 100
        self._sourness = 1
        self._sweetness = 1
        self._bitterness = 1

    def rot(self):
        self._freshness -= 1

    def get_taste(self):
        return ( self._sweetness,
                 self._sourness,
                 self._bitterness )
```

Apple Class: DIY0

```
class Apple(Fruit):

    def __init__( self, *args, **kwargs ):
        super().__init__(*args, **kwargs)
        self._sourness = 5
        self._sweetness = 20
        self._bitterness = 2

    def rot(self):
        self._freshness -= 4
```

Apple & Lemon Class

```
class Apple(Fruit):

    def __init__( self, *args, **kwargs ):
        super().__init__(*args, **kwargs)
        self._sourness = 5
        self._sweetness = 20
        self._bitterness = 2

    def rot(self):
        self._freshness -= 4
```

```
class Lemon(Fruit):

    def __init__( self, *args, **kwargs ):
        super().__init__(*args, **kwargs)
        self._sourness = 20
        self._sweetness = 2
        self._bitterness = 5

    def rot(self):
        self._bitterness += 1
        self._freshness -= 2
```

Instantiation

Class instances

```
a = Apple(20, 10)
l = Lemon(vit=30, weight=10)
l2 = Lemon(30, weight=11)
l3 = Lemon(weight=11, 30)
```

```
l3 = Lemon(weight=11, vit=30)
SyntaxError: non-keyword arg after keyword arg
```

DIY 1

Create a Nature manager which rots all available fruits until there are no fruits left

- instantiate fruits into a collection
- pass this collection of fruits to the manager
- the manager will call the rot() method of fruits
- remove fruits from the collection if it is rotten
- exit when all fruits have rotten

DIY 1

```
class Nature(object):

    def __init__(self, fruits):
        self.fruits = fruits

    def decay(self):
        to_delete = []
        for fruit in self.fruits:
            fruit.rot()
            if fruit.is_rotten:
                to_delete.append(fruit)

        for fruit in to_delete:
            print("fruit {0} has rotten".format(fruit))
            self.fruits.remove(fruit)

    def run(self):
        while len(self.fruits):
            self.decay()

fruits = [ Apple(20,20) for x in range(20) ]
for x in range(20):
    fruits.append( Lemon( 30, 10) )
Nature(fruits).run()
```

Standard methods

Syntax	Python calls
`x = MyClass()`	`__init__`
`del x`	`__del__`
`for i in x:`	`__iter__ / __next__`
`print(x)`	`__repr__`
`x()`	`__call__`
`len(x)`	`__len__`
`y in x:`	`__contains__`
`x + y`	`__add__`
`x - y`	`__sub__`
`x * y`	`__mul__`
`x == y`	`__eq__`
`x != y`	`__ne__`
`x > y`	`__lt__`
`x >= y`	`__le__`
`if x:`	`__bool__`

https://docs.python.org/3/reference/datamodel.html#basic-customization

DIY 2

Add methods to determine the fruits 'tastiness'.

Make sure you can compare the fruits taste or mix them

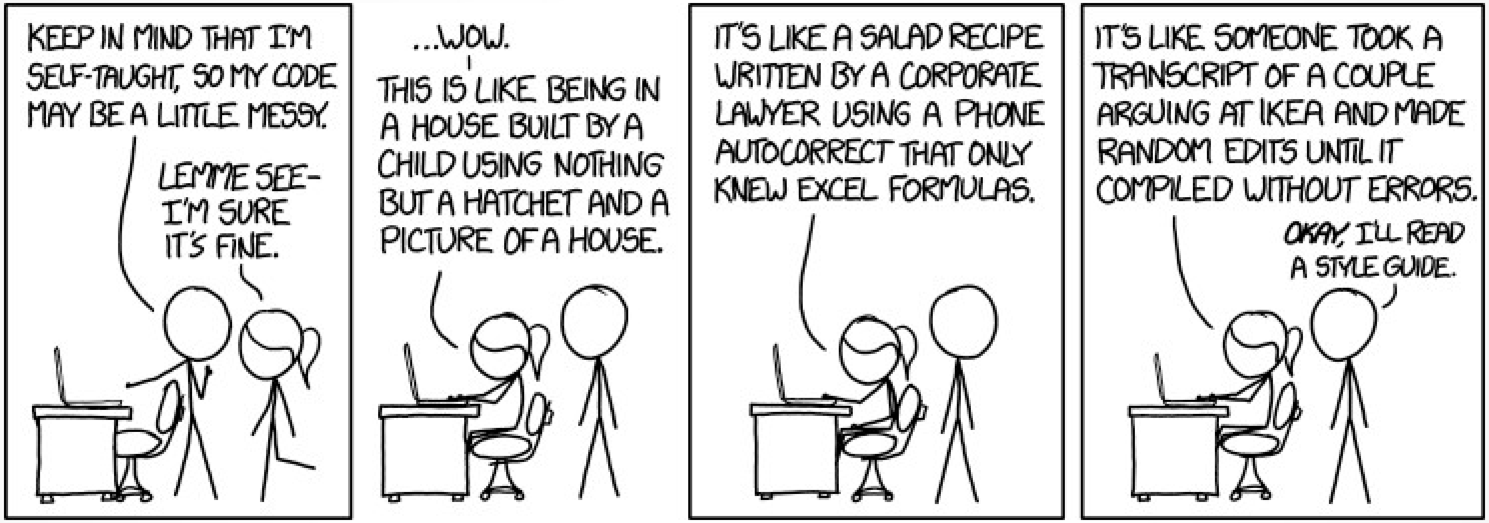
For example what taste do we get if we sum an Apple and a Lemon

```
a = Apple()
l = Lemon()
a + l
```

(multiple)Inheritance & MixIns

```
class TastyMixin(object):
    """
    This class provides taste comparison operators
    """
    def __lt__(self, other):
        st = self._sourness + self._sweetness + self._bitterness
        ot = other._sourness + other._sweetness + other._bitterness
        return st < ot

    def __le__(self, other):
        st = self._sourness + self._sweetness + self._bitterness
        ot = other._sourness + other._sweetness + other._bitterness
        return st <= ot

    def __eq__(self, other):
        st = self._sourness + self._sweetness + self._bitterness
        ot = other._sourness + other._sweetness + other._bitterness
        return st == ot

    def __ne__(self, other):...
    def __gt__(self, other):...
    def __ge__(self, other):...
```

Coding Style


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

Coding Style

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

PEP8: Python Style Guide

- Indentation: use 4 spaces (Spaces are preferred!)
- Align with opening delimiter:

```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

- multi-line constructs line up

```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]
```

- Limit all lines to a maximum of 79 characters.
- Surround top-level function and class definitions with two blank lines.
- Method definitions inside a class are surrounded by a single blank line.
- Always use UTF-8 for source files (Python3)
- For core source only use ASCII characters

http://www.python.org/dev/peps/pep-0008/

PEP8: Python Style Guide

- imports on seperare lines
- imports always at the top of a file before globals and constants
- imports should be grouped as follows
  1) standard lib imports
  2) third party lib imports
  3) local imports
- avoid wild card imports (from <module> import *)

```
import os
import sys
# No import os, sys
from subprocess import Popen, PIPE

import numpy

from myclass import MyClass
from foo.bar.yourclass import YourClass
```

http://www.python.org/dev/peps/pep-0008/

PEP8: Python Style Guide

- bad comments are worse than no comments!!!
- use inline comments sparingly
- indent block comments to the same level as the code

```
x = x + 1                 # Increment x (this is bad)
x = x + 1                 # Compensate for border

   # Use the log() method instead of print() to
   # align with debug levels
   def log(msg):
      ...
```

http://www.python.org/dev/peps/pep-0008/

PEP8: Python Style Guide

- modules should have all-lowercase names
- Class names should use CapWords
- Exceptions should use the Error suffix
- Function/method names should be lowercase with words seperated by _
- Use one leading underscore only for non-public methods and instance variables.
- Contstants all capital letters with underscores separating words
-

```
import foo.bar

BLA = 0

class CapWord(object):
    “””
    Write docstrings for all public classes
    “””

    _internal_var = 3

    def do_bla(self):
        if False:
            raise CapWordError(“d'oh”)
```

http://www.python.org/dev/peps/pep-0008/

DIY3

Install pep8

```
$ pip install pep8
```

Or through the distro package management

```
$ sudo apt-get install pep8
```

Run pep8 on your .py file containing the previous classes until no more errors

```
$ pep8 diy1.py
diy0.py:13:1: W293 blank line contains whitespace
diy0.py:22:1: W293 blank line contains whitespace
diy0.py:24:1: W293 blank line contains whitespace
diy0.py:25:18: E201 whitespace after '('
diy0.py:25:40: E202 whitespace before ')'
```

:)

Install autopep8

```
$ pip install autopep8
$ autopep8 --in-place diy1.py
```

Pythonic Thinking

> all your pseudo code are belong to us

Raw Strings

```
>>> "I'm a string,\nm'kay"
"I'm a string,\nm'kay"

>>> r"I'm a string,\nm'kay"
"I'm a string,\\nm'kay"

>>> b"I'm a string,\nm'kay"
b"I'm a string,\nm'kay"

>>> u"I'm a string,\nm'kay"
"I'm a string,\nm'kay"

>>> """
... I'm a string
...
... m'kay"""
"\nI'm a string\n\nm'kay"

>>> "I'm a string,\nm'kay".upper()
"I'M A STRING,\nM'KAY"

>>> "I like the numbers {0} and {1}".format(3, 4**2)
"I like the numbers 3 and 16"
```

Looping/Iterators

```python
a = [0, 1, 2, 3, 4, 6]
for x in range(len(a)):
    print(a[x])
```

```python
a = [0, 1, 2, 3, 4, 6]
for x in a:
    print(x)
```

```python
a = [0, 1, 2, 3, 4, 6]
for i,x in enumerate(a):
    print(i, x, a[i])
```

pythonic

Looping (dangerous)

```
a = [0, 1, 2, 3, 4, 6]
for x in range(len(a)):
    print(a[x])
    if x == 3:
        a.remove(a[x])
```

```
a = [0, 1, 2, 3, 4, 6]
for x in a:
    print(x)
    if x == 3:
        a.remove(x)
```

```
a = [0, 1, 2, 3, 4, 6]
for i,x in enumerate(a):
    print(i, x, a[i])
    if x == 3:
        a.remove(x)
```

Looping/Iterators

```
a = { 0: 1, 2: 3, 4: 6}
for k in a:
    print(k, a[k])
```

```
a = { 0: 1, 2: 3, 4: 6}
for k in a.keys():
    print(k, a[k])
```

```
a = { 0: 1, 2: 3, 4: 6}
for v in a.values():
    print(v)
```

pythonic

```
a = { 0: 1, 2: 3, 4: 6}
for k,v in a.items():
    print(k,v)
```

Looping?

```
a = { 0: 1, 2: 3, 4: 6}
for k,v in a.items():
   print(k,v)
   if k == 2:
       del a[k]
```

Looping/Iterators

```
q = ['name', 'quest', 'favorite color']
a = ['lancelot', 'the holy grail', 'blue']
for i, v in enumerate(q):
    print('What is your {0}? It is {1}.'.format(v, a[i]))
```

```
q = ['name', 'quest', 'favorite color']
a = ['lancelot', 'the holy grail', 'blue']
for v1, v2 in zip(q, a):
    print('What is your {0}? It is {1}.'.format(v1, v2))
```

Slicing

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

Slicing

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
>>> a[:1]
[0]
>>> a[:2]
[0, 1]
>>> a[:3]
[0, 1, 2]
```

Slicing

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
>>> a[3:]
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[2:]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[1:]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[0:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[-1:]
```

Slicing

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
>>> a[3:]
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[2:]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[1:]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[0:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[-1:]
[15]
```

Slicing

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
>>> a[3:8]
[3, 4, 5, 6, 7]
>>> a[-10:-1]
[6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> a[6:15]
[6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Slicing + strides

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
>>> a[6:15:2]
[6, 8, 10, 12, 14]
>>> a[6:15:3]
[6, 9, 12]
>>> a[6::3]
[6, 9, 12, 15]
>>> a[::3]
[0, 3, 6, 9, 12, 15]
```

Slicing + strides

```
>>> for v in a[::2]: v & 1
...
0
0
0
0
0
0
0
0
>>> for v in a[::2]: bool(v & 1)
...
False
False
False
False
False
False
False
False
>>> for v in a[1::2]: bool(v & 1)
...
True
```

Exceptions

```
cupcakes = 9

def yo_mama():
    global cupcakes
    cupcakes -= 1
    if cupcakes:
        return cupcakes
    # implicit return None

for x in range(9):
    print(yo_mama())
```

Exceptions

```
def yo_mama2():
    global cupcakes
    cupcakes -= 1
    if cupcakes < 3:
        return cupcakes, "We need moa"
    elif cupcakes:
        return cupcakes
    # implicit return None

cupcakes = 9
for x in range(9):
    print(yo_mama2())
```

Exceptions

```
class OutOfCupCakesError(Exception):
    """We're out of cupcakes"""

def yo_mama3():
    global cupcakes
    cupcakes -= 1
    if cupcakes < 3:
        raise OutOfCupCakesError
    elif cupcakes:
        return cupcakes
    # implicit return None

cupcakes = 9
for x in range(9):
    try:
        print(yo_mama3())
    except OutOfCupCakesError:
        print("we need moa")
```

Exceptions

```
try:
    do_something()
    do_anotherthing()
except SomeError as e:
    print(e)
else:
    print("Jay, we managed")
finally:
    print("Die peacefully")
```

Exceptions

```
try:
    do_something()
    do_anotherthing()
except (SomeError, AnotherError) as e:
    print(e)
except:
    print("I give up")
else:
    print("Jay, we managed")
finally:
    print("Die peacefully")
```

DIY4

Add exceptions to the fruit classes and nature managers

make the nature fruit class iterable by taste?

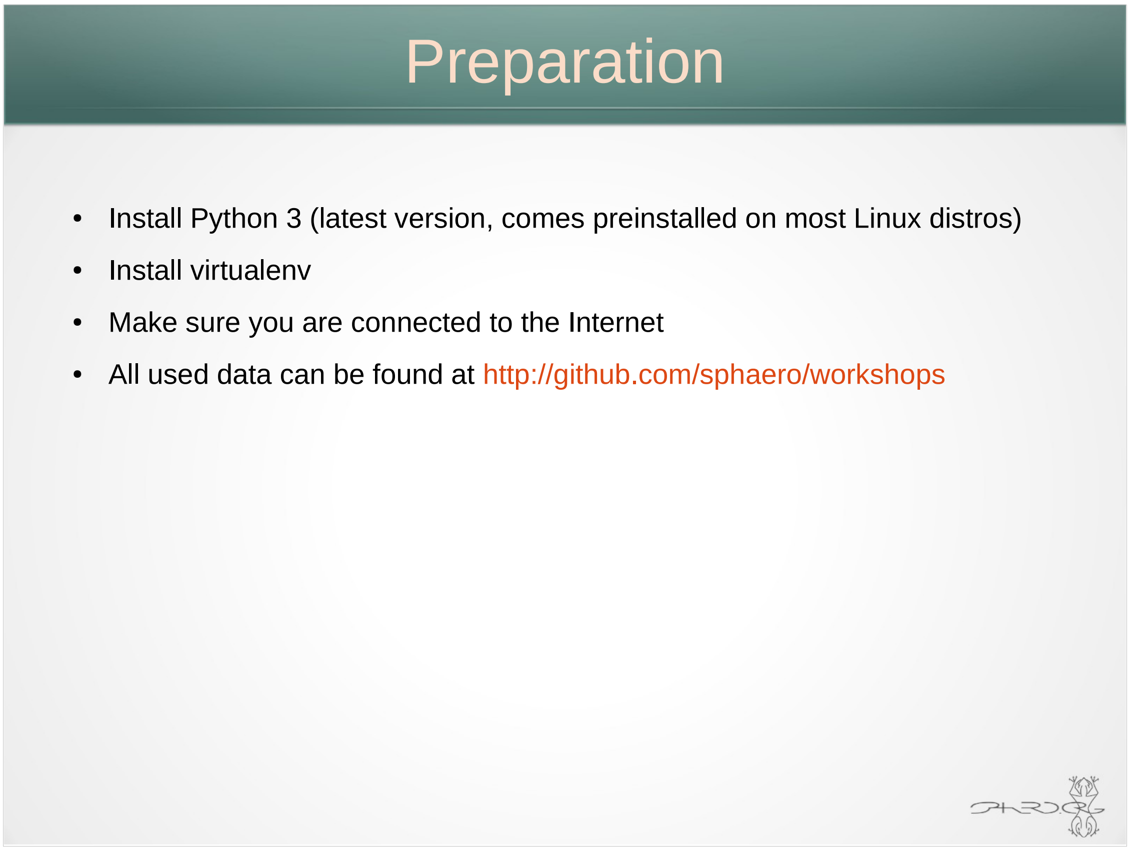
or anything you picked up today

sys.exit(0)

PythonFu

Arnaud Loonstra, 12-08-2016, Amsterdam

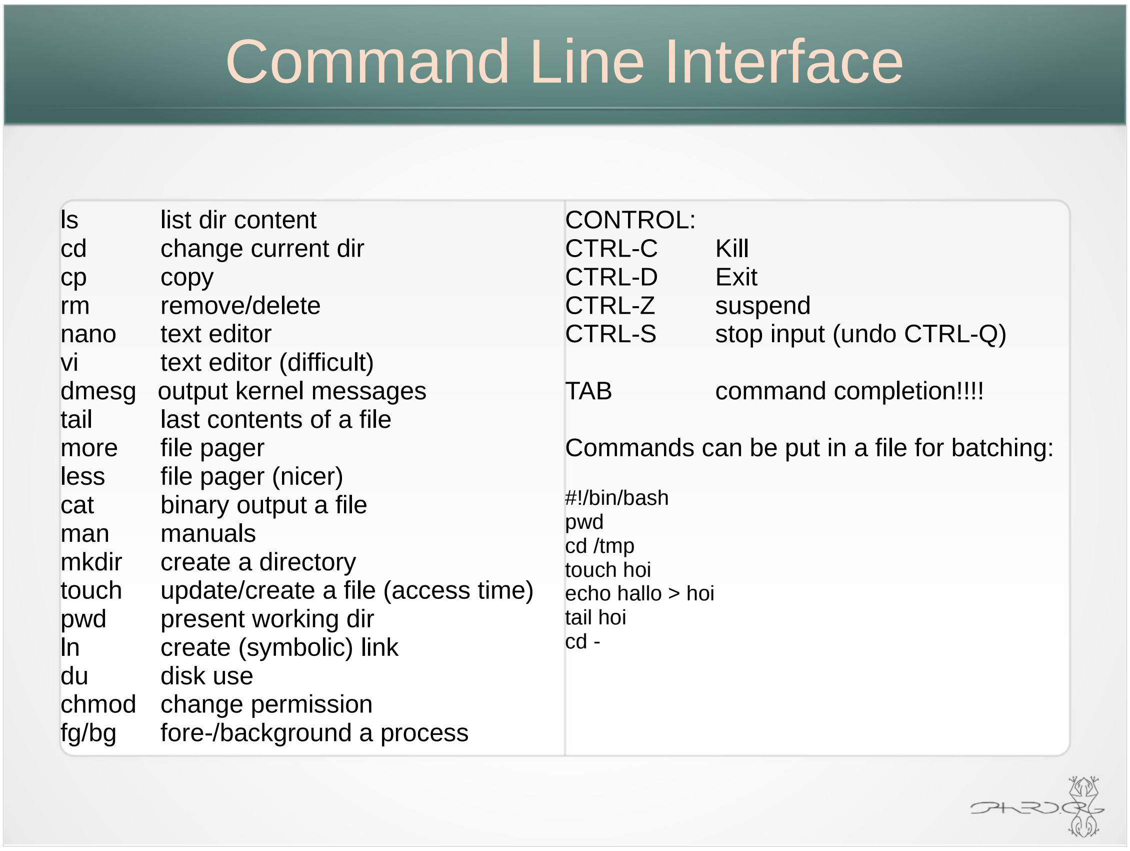
Welkom voorstel etc



Preparation

- Install Python 3 (latest version, comes preinstalled on most Linux distros)
- Install virtualenv
- Make sure you are connected to the Internet
- All used data can be found at http://github.com/sphaero/workshops

Favoriete texteditor paraat hebben!!!



Command Line Interface

ls	list dir content
cd	change current dir
cp	copy
rm	remove/delete
nano	text editor
vi	text editor (difficult)
dmesg	output kernel messages
tail	last contents of a file
more	file pager
less	file pager (nicer)
cat	binary output a file
man	manuals
mkdir	create a directory
touch	update/create a file (access time)
pwd	present working dir
ln	create (symbolic) link
du	disk use
chmod	change permission
fg/bg	fore-/background a process

CONTROL:

CTRL-C	Kill
CTRL-D	Exit
CTRL-Z	suspend
CTRL-S	stop input (undo CTRL-Q)
TAB	command completion!!!!

Commands can be put in a file for batching:

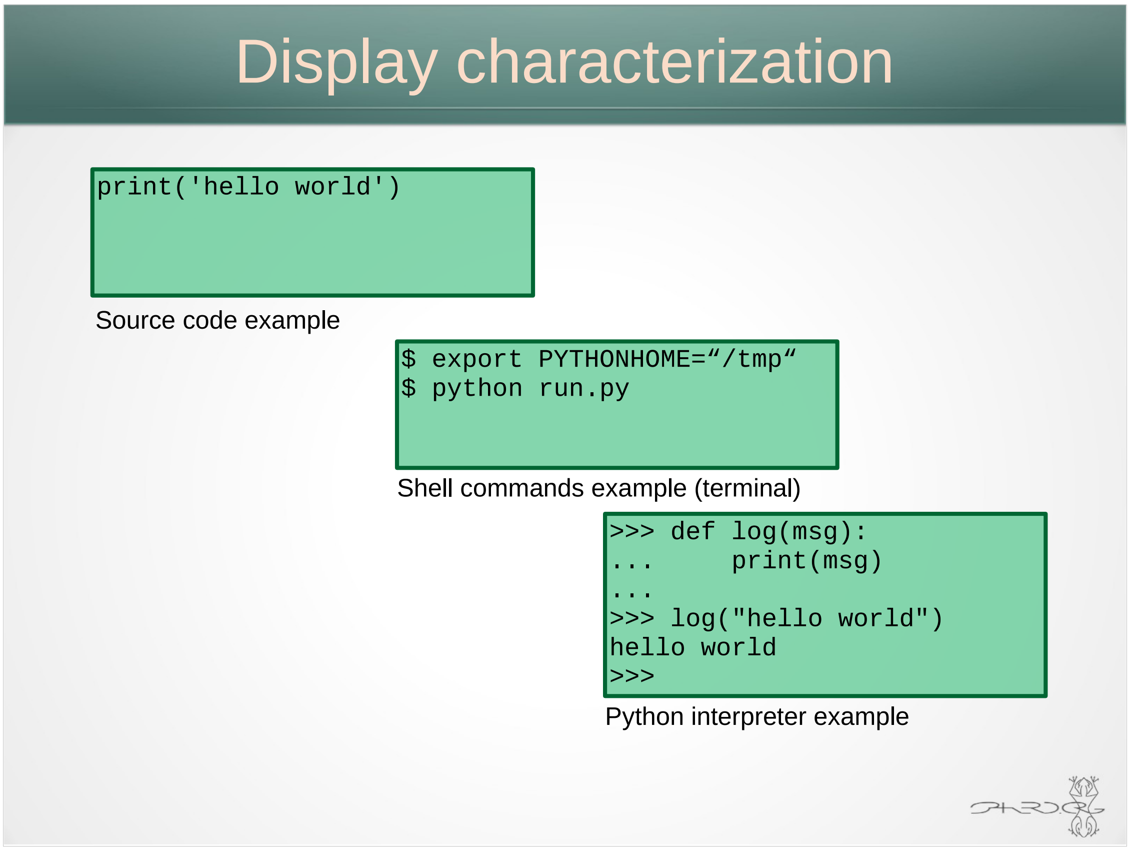
```
#!/bin/bash
pwd
cd /tmp
touch hoi
echo hallo > hoi
tail hoi
cd -
```

We'll be using CLI in Linux

'under the hood'

Sheet with all commands, keep track

Ctrl-C to stop
TAB to complete



Display characterization

```
print('hello world')
```

Source code example

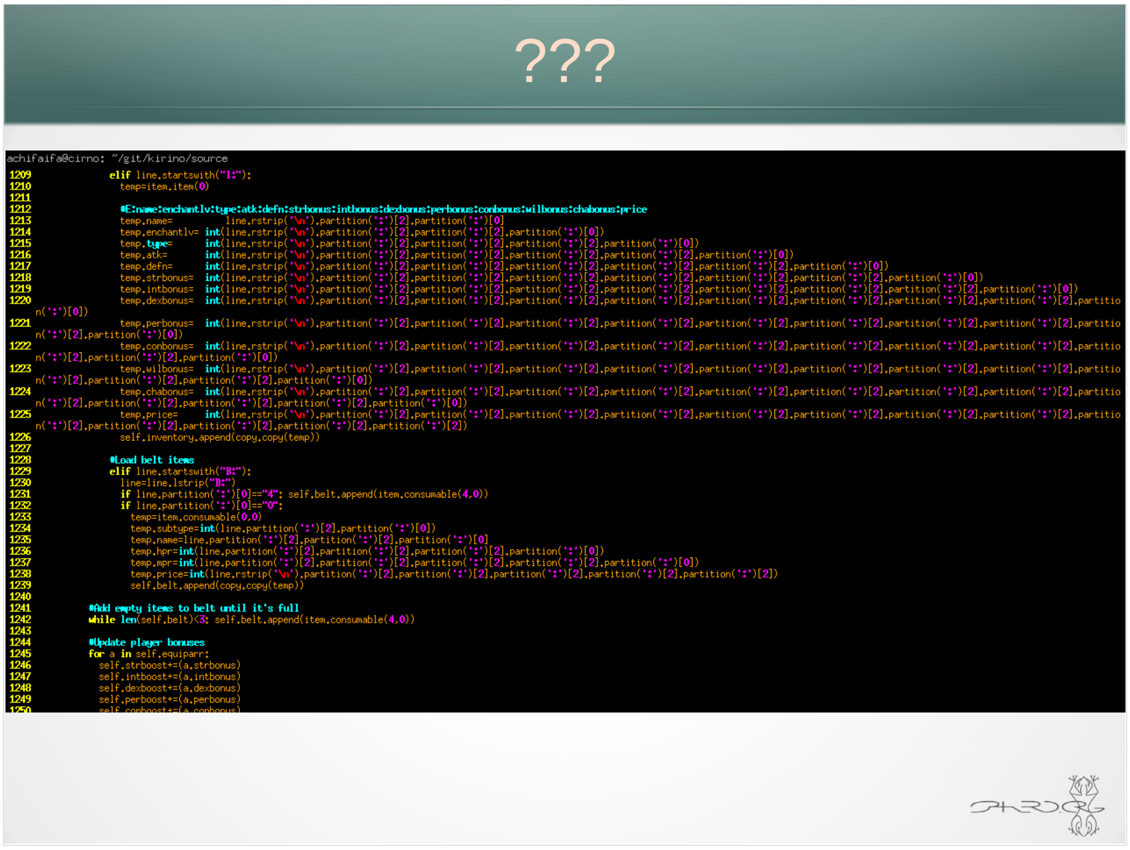
```
$ export PYTHONHOME=“/tmp“
$ python run.py
```

Shell commands example (terminal)

```
>>> def log(msg):
...     print(msg)
...
>>> log("hello world")
hello world
>>>
```

Python interpreter example

Uitleggen van voorbeeld code in sheets



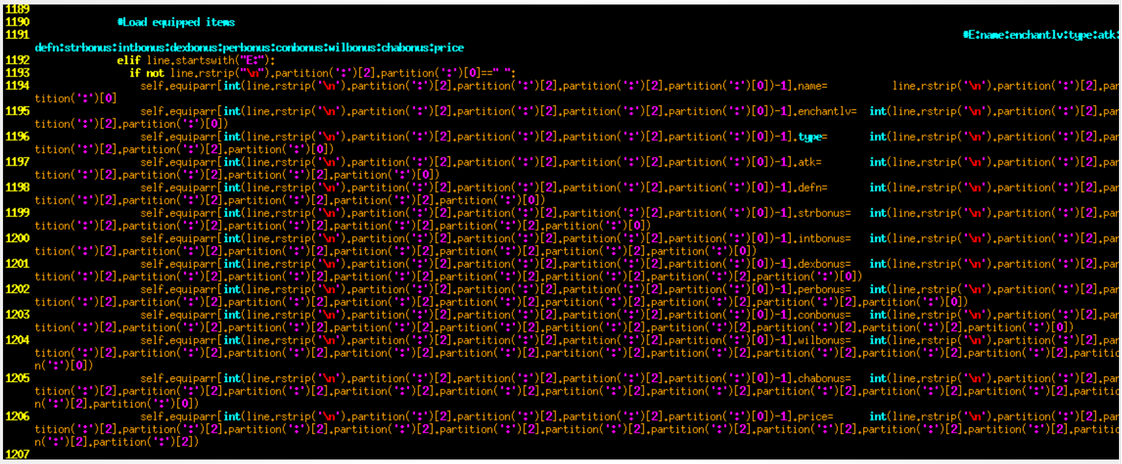
???

```
achifaifa@cirno: ~/git/kirino/source
              elif line.startswith("I:"):
                temp=item.item(0)

                #E:name:enchantlv:type:atk:defn:strbonus:intbonus:dexbonus:perbonus:conbonus:wilbonus:chabonus:price
                temp.name=          line.rstrip('\n').partition(':')[2].partition(':')[0]
                temp.enchantlv= int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[0])
                temp.type=      int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.atk=       int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.defn=      int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.strbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.intbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.dexbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[0])
                temp.perbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[2].partition(':')[0])
                temp.conbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[2].partition(':')[2].partition(':')[0])
                temp.wilbonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.chabonus=  int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                temp.price=     int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2])
                self.inventory.append(copy.copy(temp))

              #Load belt items
              elif line.startswith("B:"):
                line=line.lstrip("B:")
                if line.partition(':')[0]=="4": self.belt.append(item.consumable(4,0))
                if line.partition(':')[0]=="0":
                  temp=item.consumable(0,0)
                  temp.subtype=int(line.partition(':')[2].partition(':')[0])
                  temp.name=line.partition(':')[2].partition(':')[2].partition(':')[0]
                  temp.hpr=int(line.partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  temp.mpr=int(line.partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  temp.price=int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2])
                  self.belt.append(copy.copy(temp))

          #Add empty items to belt until it's full
          while len(self.belt)<3: self.belt.append(item.consumable(4,0))

          #Update player bonuses
          for a in self.equiparr:
            self.strboost+=(a.strbonus)
            self.intboost+=(a.intbonus)
            self.dexboost+=(a.dexbonus)
            self.perboost+=(a.perbonus)
            self.conboost+=(a.conbonus)
```

FOSDEM vorig jaar
Ik snap dit niet... Ik weet niet wat jullie al weten en wat jullie willen weten dus kom met vragen

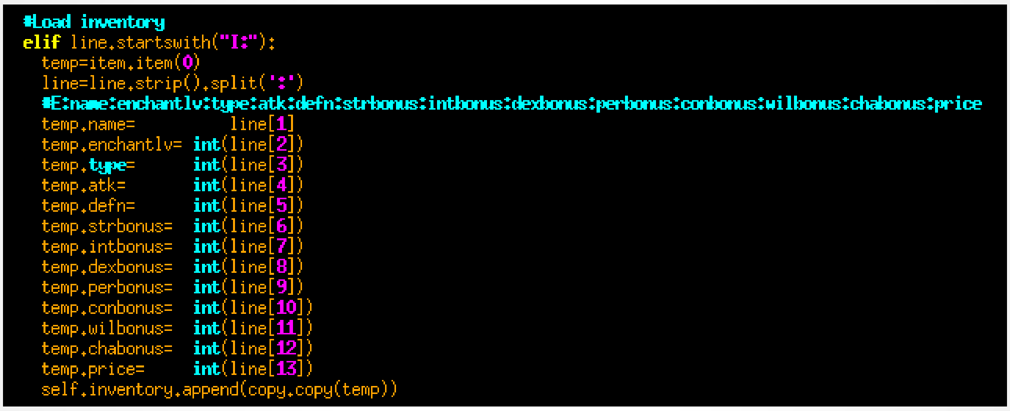
Ik heb een aantal zaken voorbereid maar ik kan er net zo goed naast zitten qua jullie behoeften

???



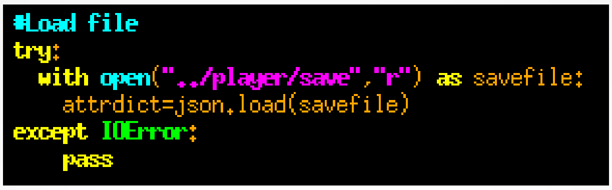
```

              #Load equipped items
                                                                                                                                                              #E:name:enchantlv:type:atk:
defn:strbonus:intbonus:dexbonus:perbonus:conbonus:wilbonus:chabonus:price
              elif line.startswith("E:"):
                if not line.rstrip("\n").partition(':')[2].partition(':')[0]==" ":
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].name=           line.rstrip('\n').partition(':')[2].par
tition(':')[0]
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].enchantlv=  int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].type=       int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].atk=        int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].defn=       int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].strbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].intbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].dexbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].perbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].conbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].wilbonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].chabonus=   int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[2].partition(':')[0])
                  self.equiparr[int(line.rstrip('\n').partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[0])-1].price=      int(line.rstrip('\n').partition(':')[2].par
tition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partition(':')[2].partitio
n(':')[2].partition(':')[2])

```

???



```
#Load inventory
elif line.startswith("I:"):
  temp=item.item(0)
  line=line.strip().split(':')
  #E:name:enchantlv:type:atk:defn:strbonus:intbonus:dexbonus:perbonus:conbonus:wilbonus:chabonus:price
  temp.name=          line[1]
  temp.enchantlv= int(line[2])
  temp.type=      int(line[3])
  temp.atk=       int(line[4])
  temp.defn=      int(line[5])
  temp.strbonus=  int(line[6])
  temp.intbonus=  int(line[7])
  temp.dexbonus=  int(line[8])
  temp.perbonus=  int(line[9])
  temp.conbonus=  int(line[10])
  temp.wilbonus=  int(line[11])
  temp.chabonus=  int(line[12])
  temp.price=     int(line[13])
  self.inventory.append(copy.copy(temp))
```

Why did this happen?

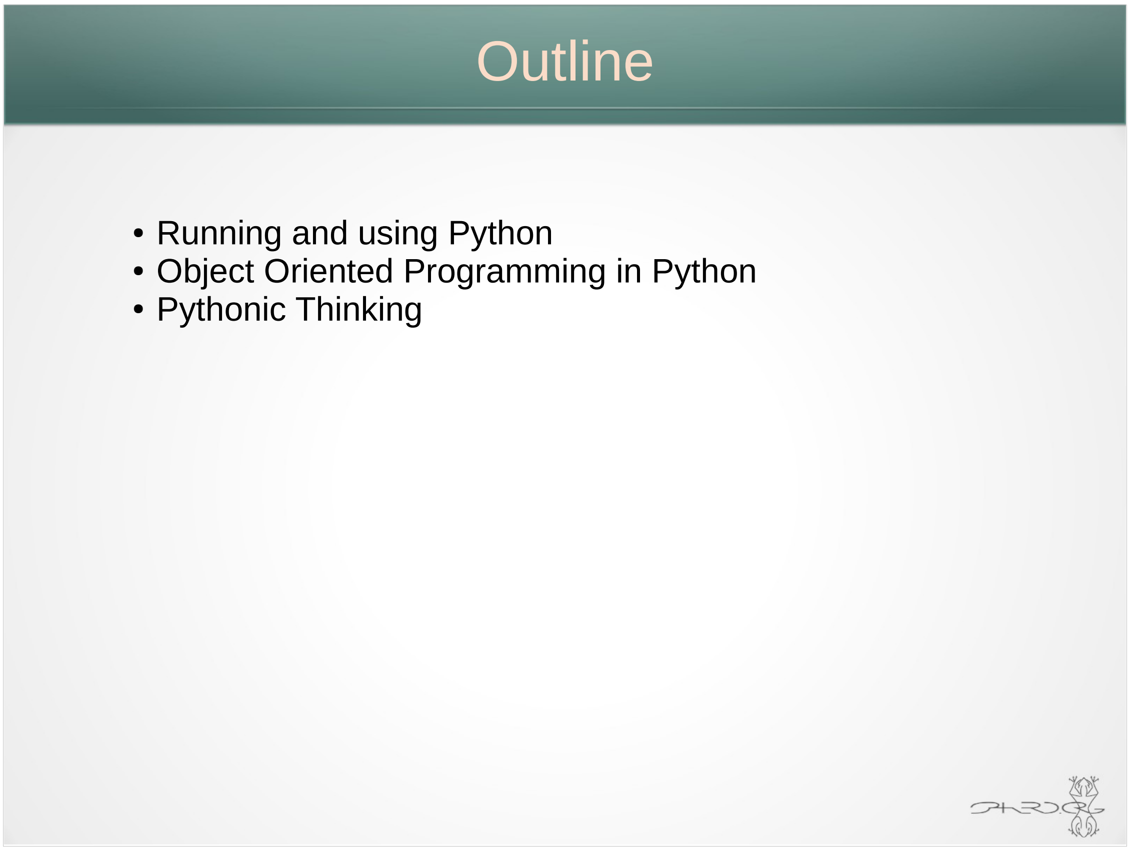


```
#Load file
try:
  with open("../player/save","r") as savefile:
    attrdict=json.load(savefile)
except IOError:
    pass
```

- The documentation lists the functions in alphabetical order
- partition() is listed before split()
- I stopped reading as soon as I found the first one

“Yuri Numerov @FOSDEM16”

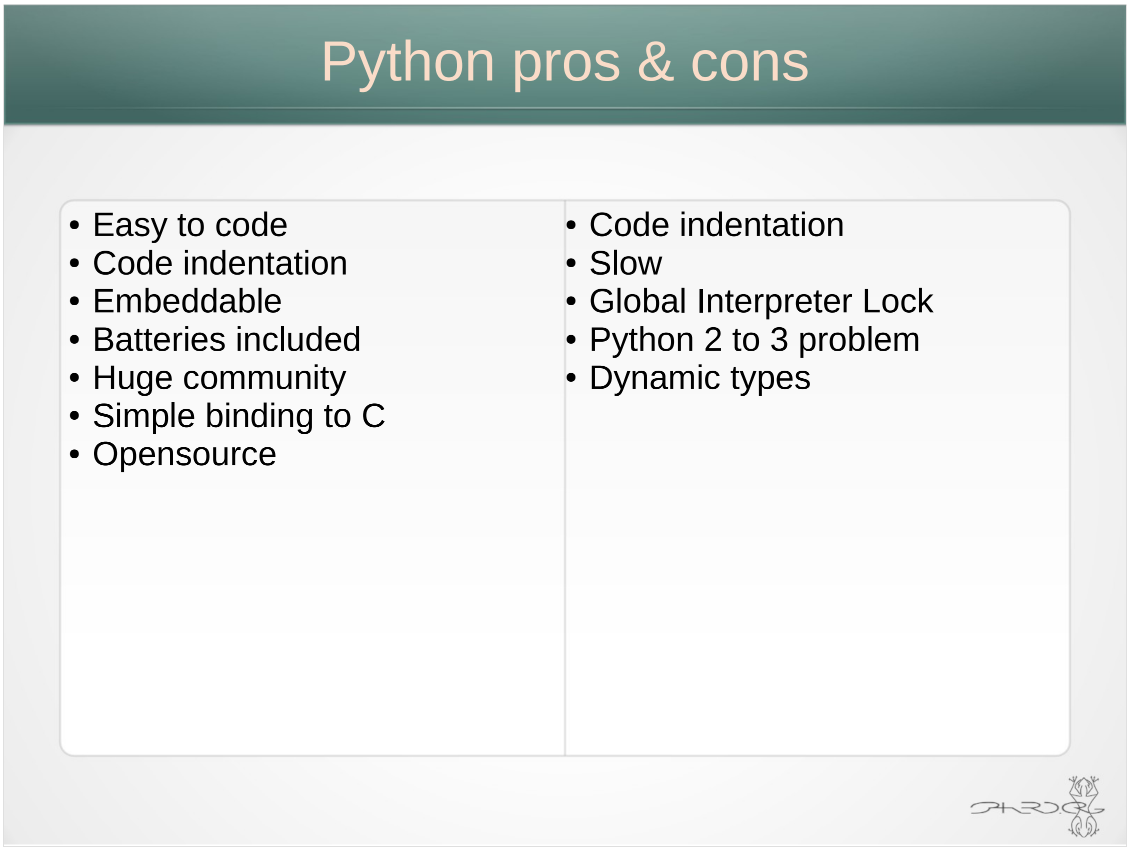
https://archive.fosdem.org/2016/schedule/event/python_mistakes/



Outline

- Running and using Python
- Object Oriented Programming in Python
- Pythonic Thinking

3 uur, */50min pauze



Python pros & cons

- Easy to code
- Code indentation
- Embeddable
- Batteries included
- Huge community
- Simple binding to C
- Opensource

- Code indentation
- Slow
- Global Interpreter Lock
- Python 2 to 3 problem
- Dynamic types

Pros & cons (eigen mening)

Python version

On unix variants:

```
$ python --version
```

On unix variants with 2 and 3 installed

```
$ python3 --version
```

On windows:

```
$ python.exe --version
```

Python version

Within Python:

```
>>> import sys
>>> sys.version
'3.4.3+ (default, Oct 14 2015, 16:03:50) \n[GCC 5.2.1
20151010]'
>>> sys.version_info
sys.version_info(major=3, minor=4, micro=3,
releaselevel='final', serial=0)
>>> sys.version_info.major
3
>>> sys.version_info.minor
4
```

Python directory structure

./include	headers for C compilers
./lib/python-wheels	wheels will replace eggs
./lib/python3.4	this is where your modules reside
./lib/python3.4/__pycache__	bytecode versions of python code (ignore)
./lib/python3.4/json	the included json module (example)
./lib/python3.4/site-packages	extra installed non native modules
./bin	the python executables

Python module lookup (path)

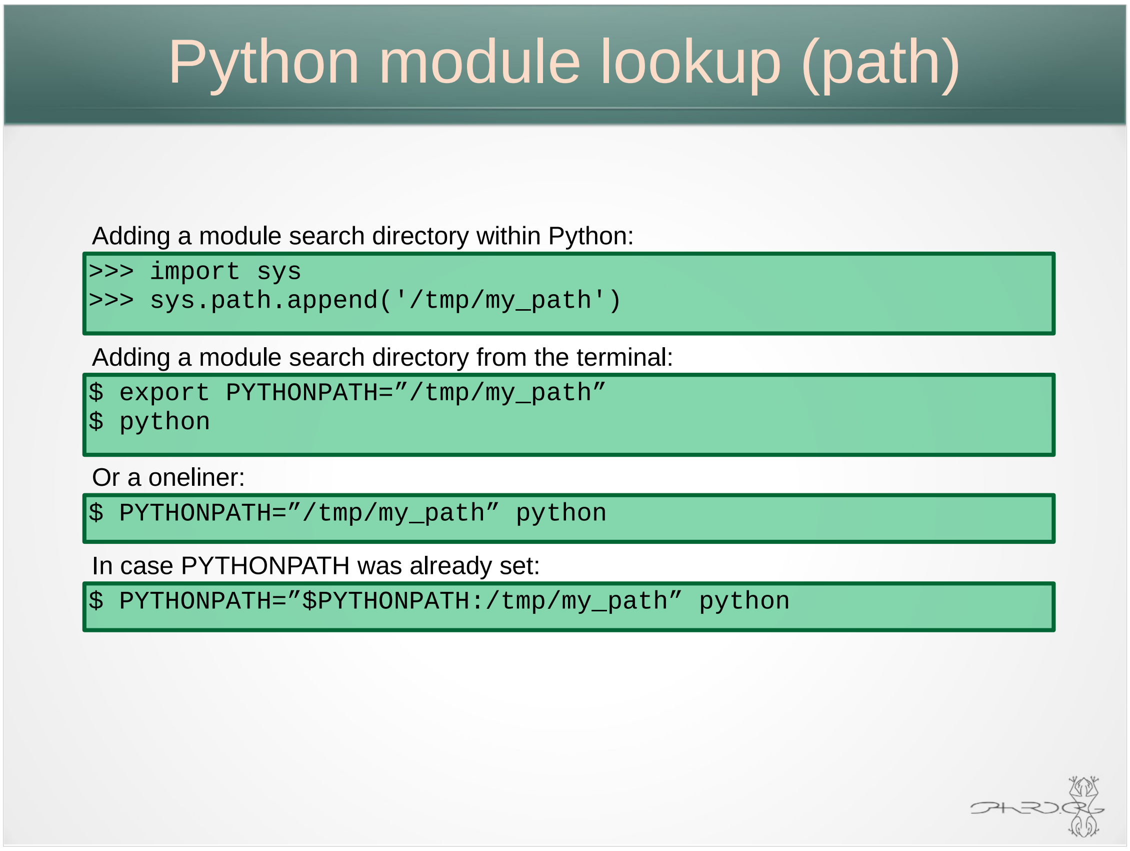
From within Python

```
>>> import sys
>>> sys.path
['', '/usr/local/lib/python3.4/dist-packages/python_axolotl-
0.1.35-py3.4.egg', '/usr/local/lib/python3.4/dist-
packages/protobuf-3.0.0b2.post2-py3.4.egg',
'/usr/local/lib/python3.4/dist-packages/pycrypto-2.6.1-py3.4-
linux-x86_64.egg', '/usr/lib/python3/dist-packages',
'/usr/lib/python3.4', '/usr/lib/python3.4/plat-x86_64-linux-
gnu', '/usr/lib/python3.4/lib-dynload',
'/home/aloonstra/.local/lib/python3.4/site-packages',
'/usr/local/lib/python3.4/dist-packages']
```

```
>>> print("\n".join(sys.path))
```

Python module lookup (path)

```
>>> print("\n".join(sys.path))

/usr/local/lib/python3.4/dist-packages/python_axolotl-0.1.35-
py3.4.egg
/usr/local/lib/python3.4/dist-packages/protobuf-3.0.0b2.post2-
py3.4.egg
/usr/local/lib/python3.4/dist-packages/pycrypto-2.6.1-py3.4-
linux-x86_64.egg
/usr/lib/python3/dist-packages
/usr/lib/python3.4
/usr/lib/python3.4/plat-x86_64-linux-gnu
/usr/lib/python3.4/lib-dynload
/home/aloonstra/.local/lib/python3.4/site-packages
/usr/local/lib/python3.4/dist-packages
```



Python module lookup (path)

Adding a module search directory within Python:

```
>>> import sys
>>> sys.path.append('/tmp/my_path')
```

Adding a module search directory from the terminal:

```
$ export PYTHONPATH=”/tmp/my_path”
$ python
```

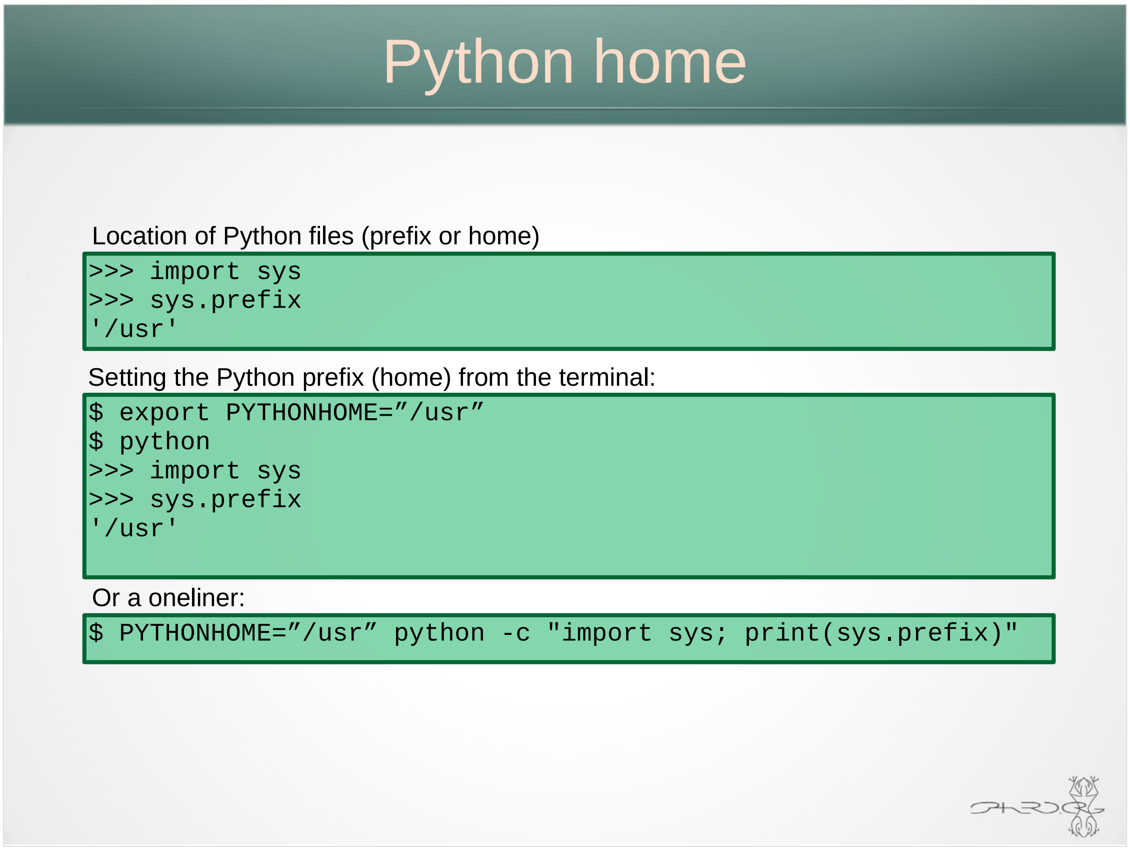
Or a oneliner:

```
$ PYTHONPATH=”/tmp/my_path” python
```

In case PYTHONPATH was already set:

```
$ PYTHONPATH=”$PYTHONPATH:/tmp/my_path” python
```

Installeren van modules pip of apt-get??? waar komen ze terecht



Python home

Location of Python files (prefix or home)

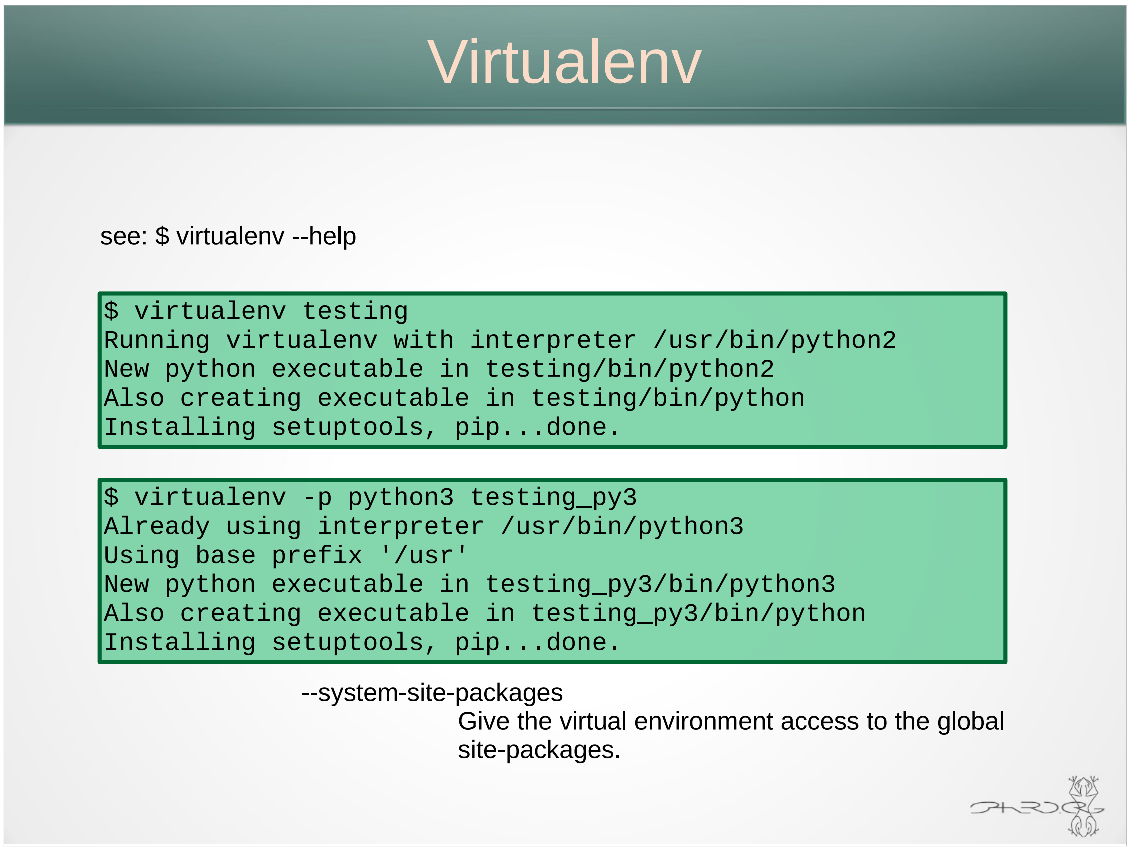
```
>>> import sys
>>> sys.prefix
'/usr'
```

Setting the Python prefix (home) from the terminal:

```
$ export PYTHONHOME=”/usr”
$ python
>>> import sys
>>> sys.prefix
'/usr'
```

Or a oneliner:

```
$ PYTHONHOME=”/usr” python -c "import sys; print(sys.prefix)"
```



Virtualenv

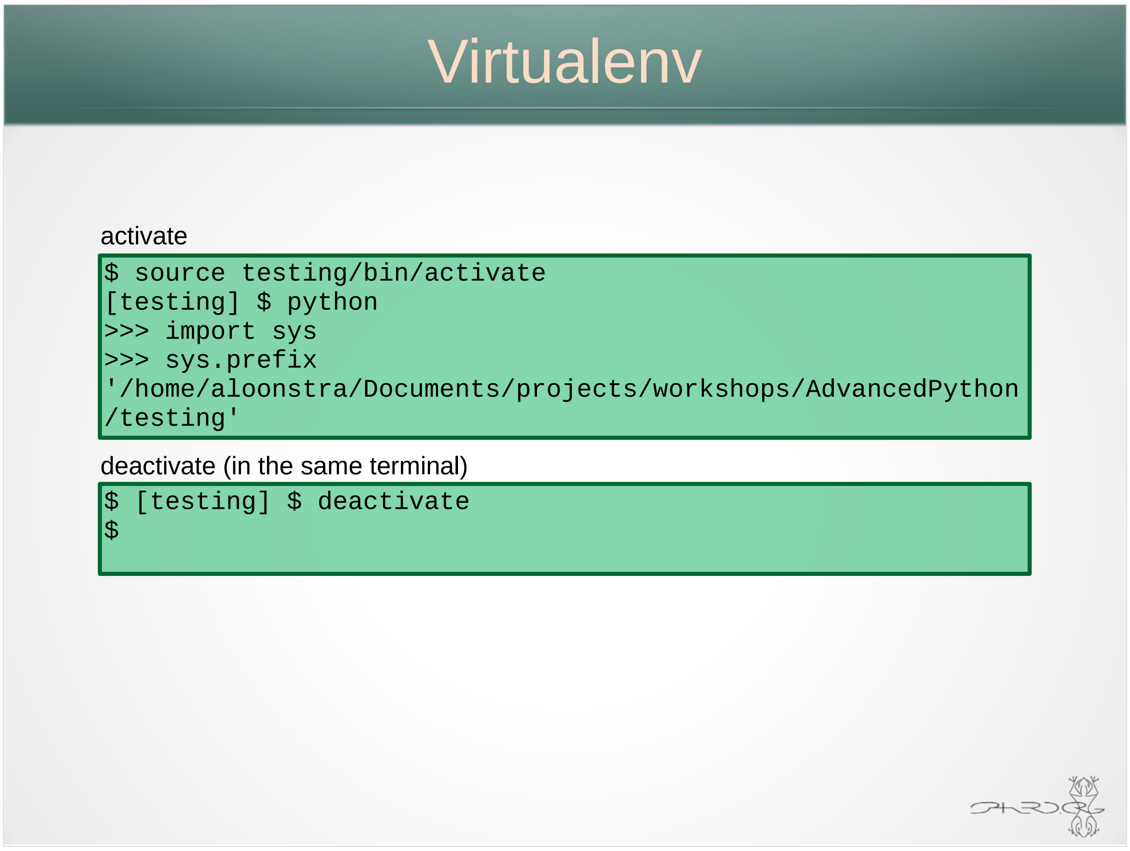
see: $ virtualenv --help

```
$ virtualenv testing
Running virtualenv with interpreter /usr/bin/python2
New python executable in testing/bin/python2
Also creating executable in testing/bin/python
Installing setuptools, pip...done.
```

```
$ virtualenv -p python3 testing_py3
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in testing_py3/bin/python3
Also creating executable in testing_py3/bin/python
Installing setuptools, pip...done.
```

```
--system-site-packages
                    Give the virtual environment access to the global
                    site-packages.
```

$ PYTHONPATH="/usr/lib/python2.7/dist-packages/" PYTHONHOME=/home/aloonstra/Documents/projects/workshops/AdvancedPython/test/test2/ gimp



Virtualenv

activate

```
$ source testing/bin/activate
[testing] $ python
>>> import sys
>>> sys.prefix
'/home/aloonstra/Documents/projects/workshops/AdvancedPython
/testing'
```

deactivate (in the same terminal)

```
$ [testing] $ deactivate
$
```

$ PYTHONPATH="/usr/lib/python2.7/dist-packages/" PYTHONHOME=/home/aloonstra/Documents/projects/workshops/AdvancedPython/test/test2/ gimp

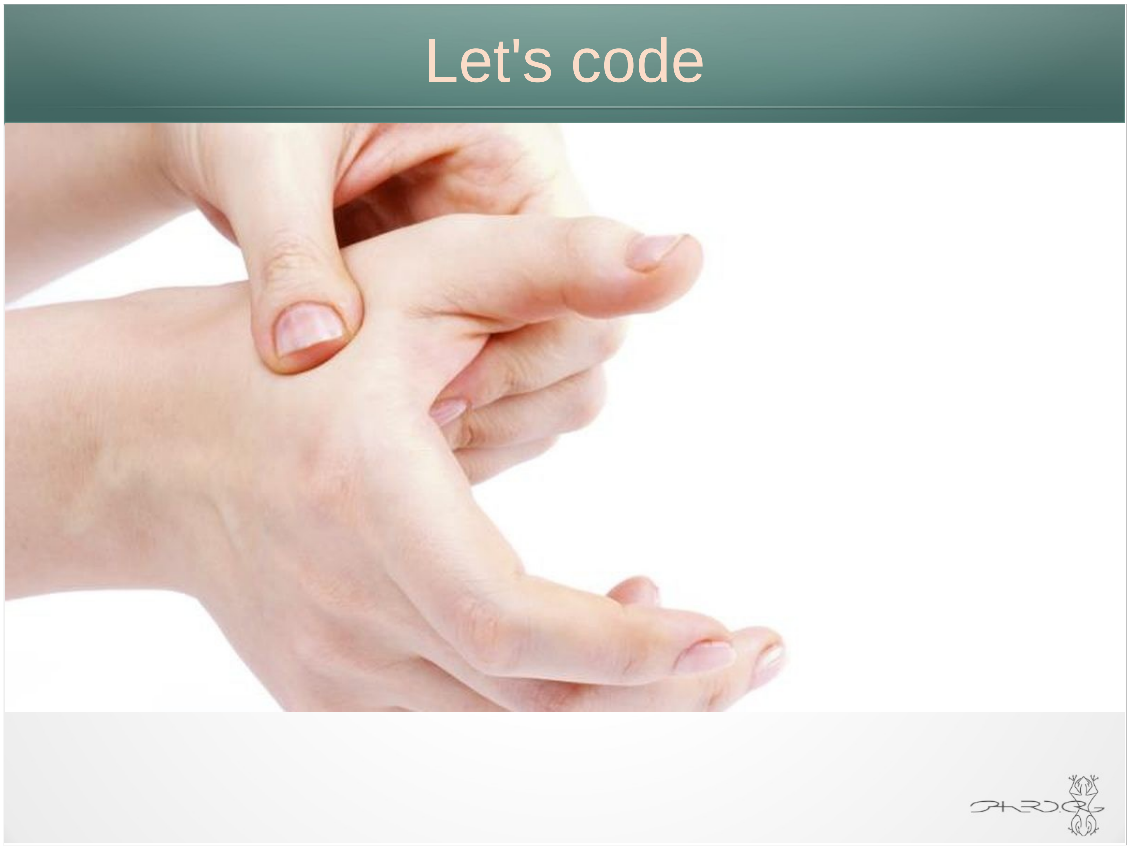
shebang

Under {Unix}, if the first two bytes of an {executable} file are "**#!**", the {kernel} treats the file as a script rather than a {machine code} program. The word following the "!" (i.e., everything up to the first {whitespace}) is used as the {pathname} of the {interpreter}.

```
#!/usr/bin/python

import ...
```

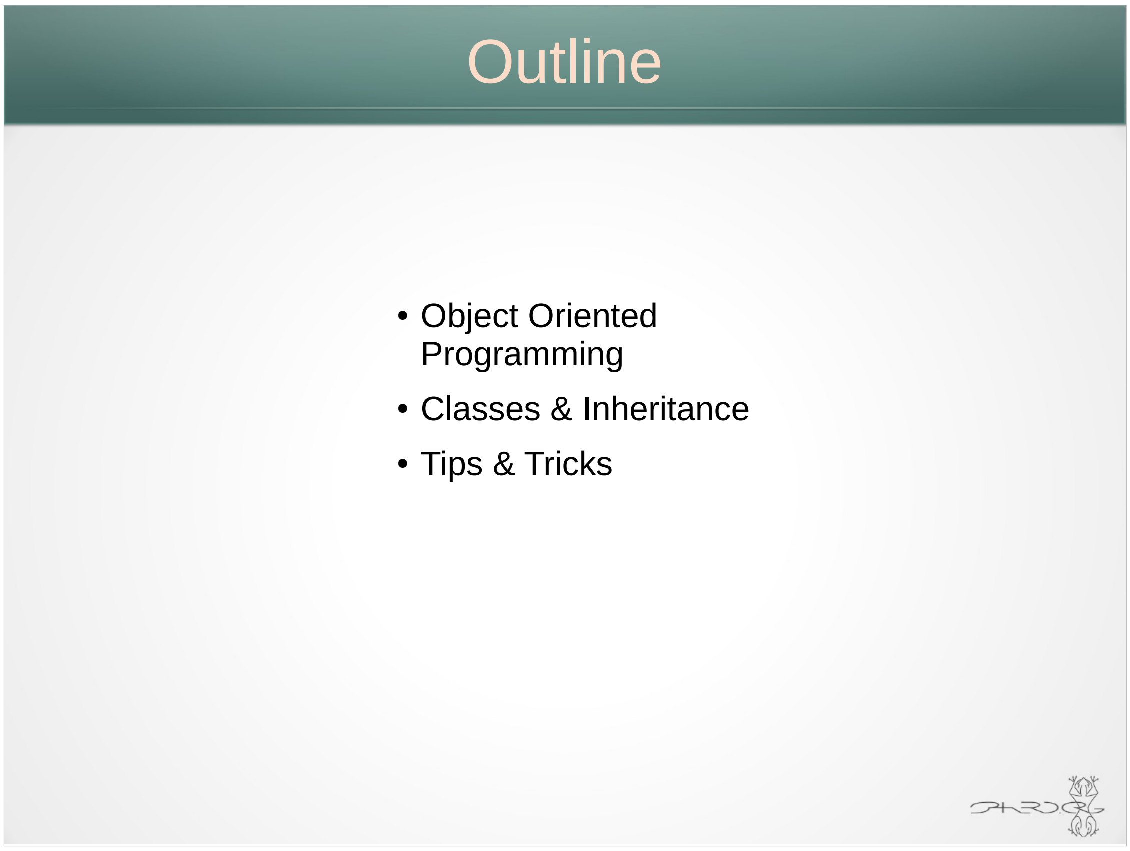
better

```
#!/usr/bin/env python

import ...
```



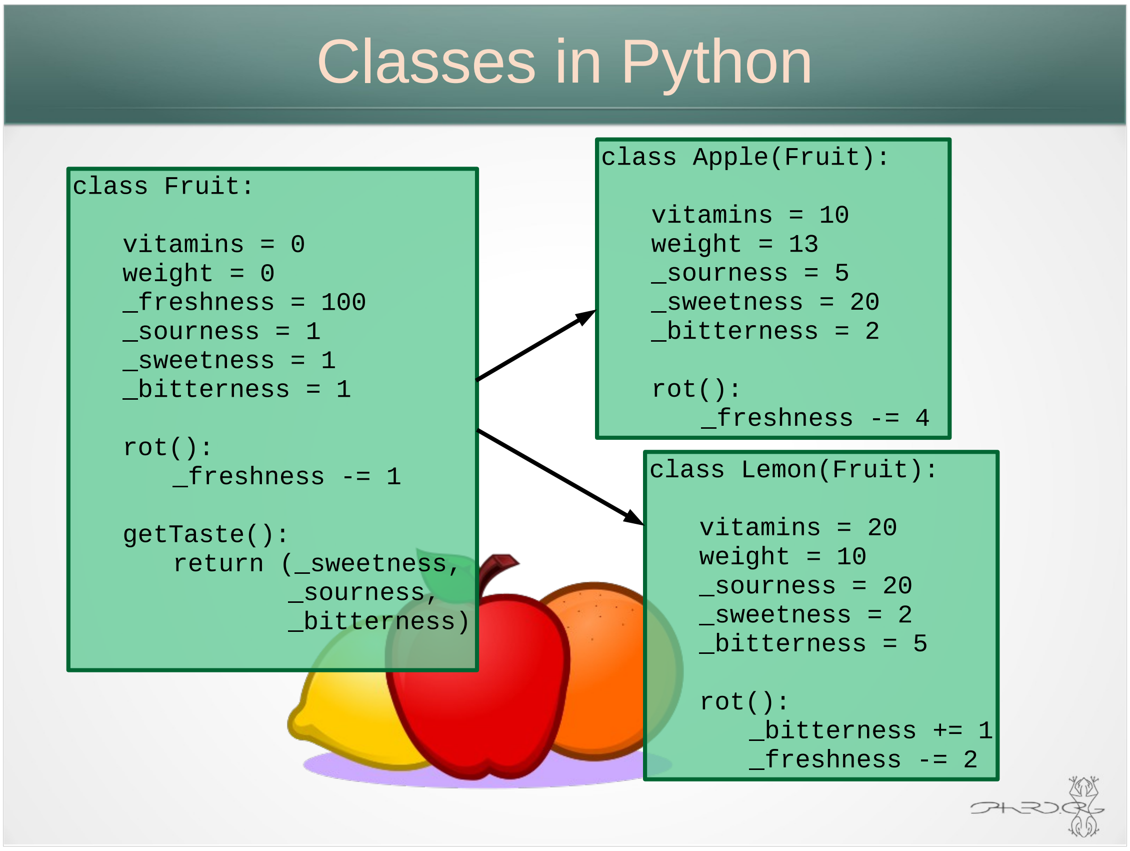
Let's code

Koffie



Outline

- Object Oriented Programming
- Classes & Inheritance
- Tips & Tricks

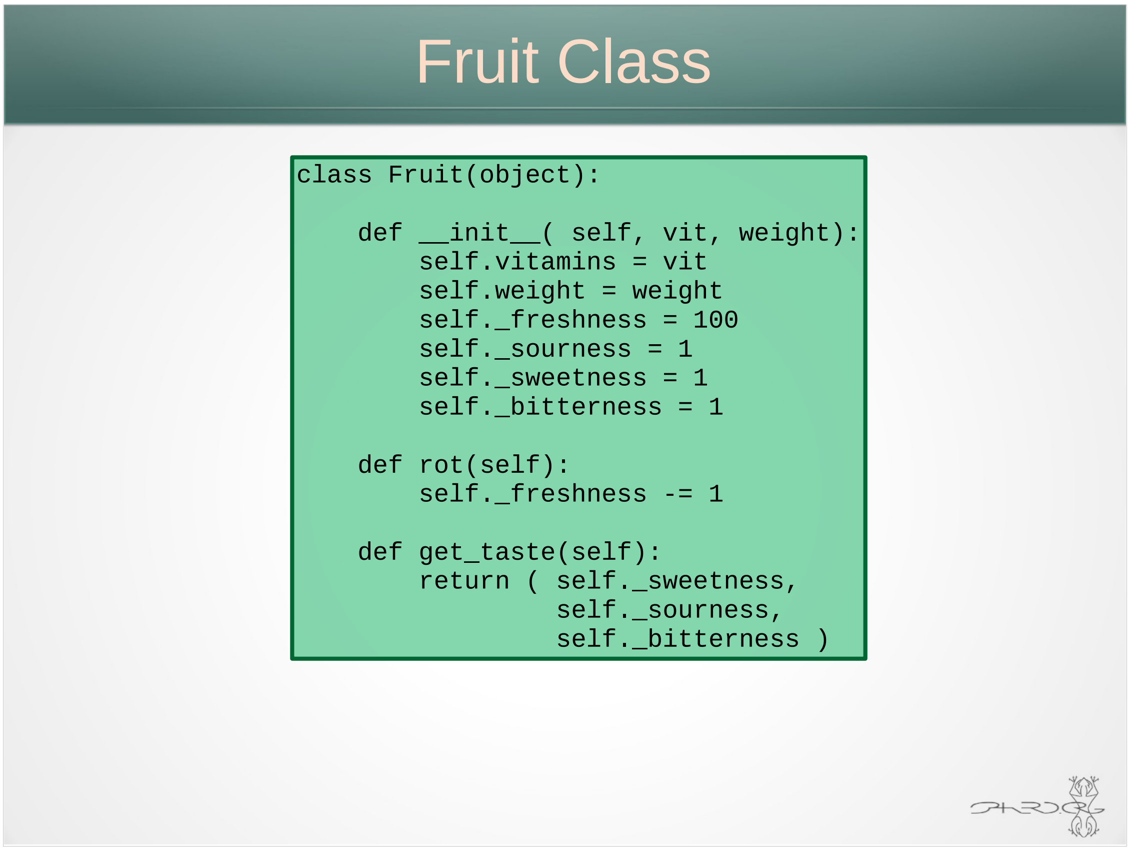


Classes in Python

```
class Fruit:

    vitamins = 0
    weight = 0
    _freshness = 100
    _sourness = 1
    _sweetness = 1
    _bitterness = 1

    rot():
        _freshness -= 1

    getTaste():
        return (_sweetness,
                _sourness,
                _bitterness)
```

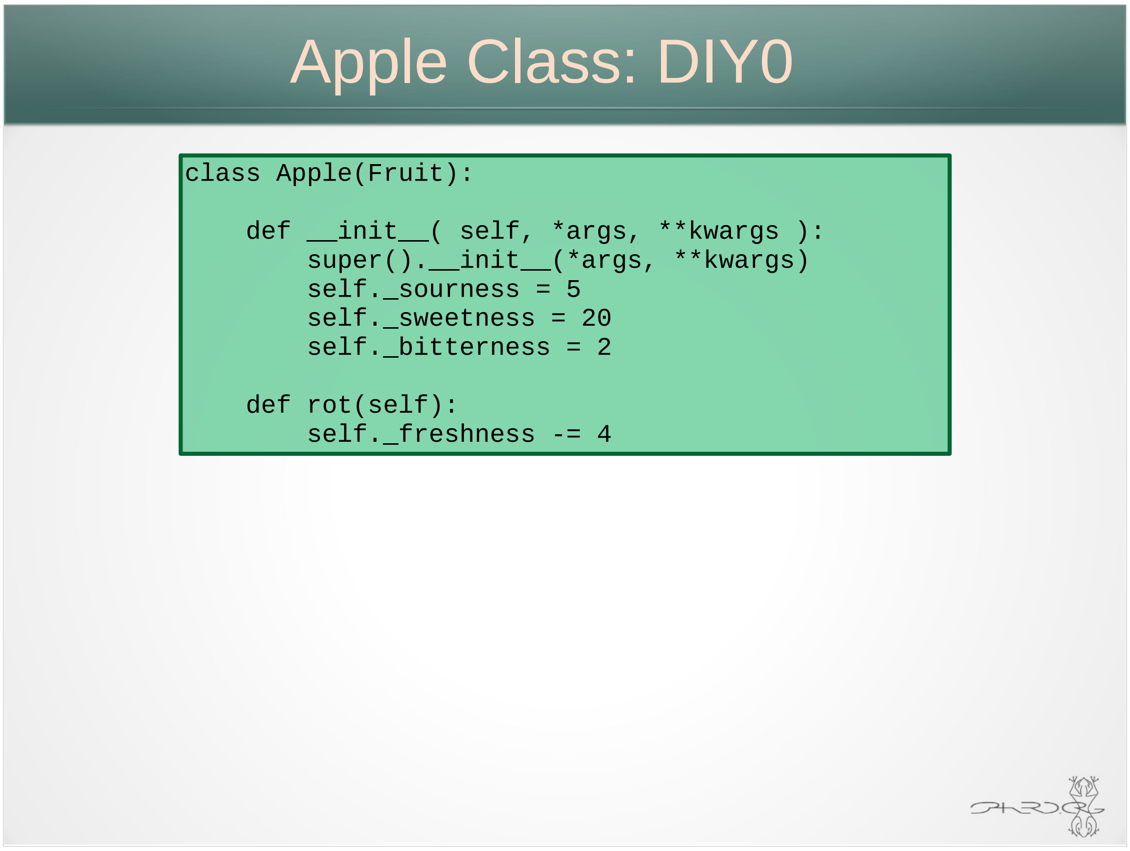
```
class Apple(Fruit):

    vitamins = 10
    weight = 13
    _sourness = 5
    _sweetness = 20
    _bitterness = 2

    rot():
        _freshness -= 4
```

```
class Lemon(Fruit):

    vitamins = 20
    weight = 10
    _sourness = 20
    _sweetness = 2
    _bitterness = 5

    rot():
        _bitterness += 1
        _freshness -= 2
```



Fruit Class

```
class Fruit(object):

    def __init__( self, vit, weight):
        self.vitamins = vit
        self.weight = weight
        self._freshness = 100
        self._sourness = 1
        self._sweetness = 1
        self._bitterness = 1

    def rot(self):
        self._freshness -= 1

    def get_taste(self):
        return ( self._sweetness,
                 self._sourness,
                 self._bitterness )
```

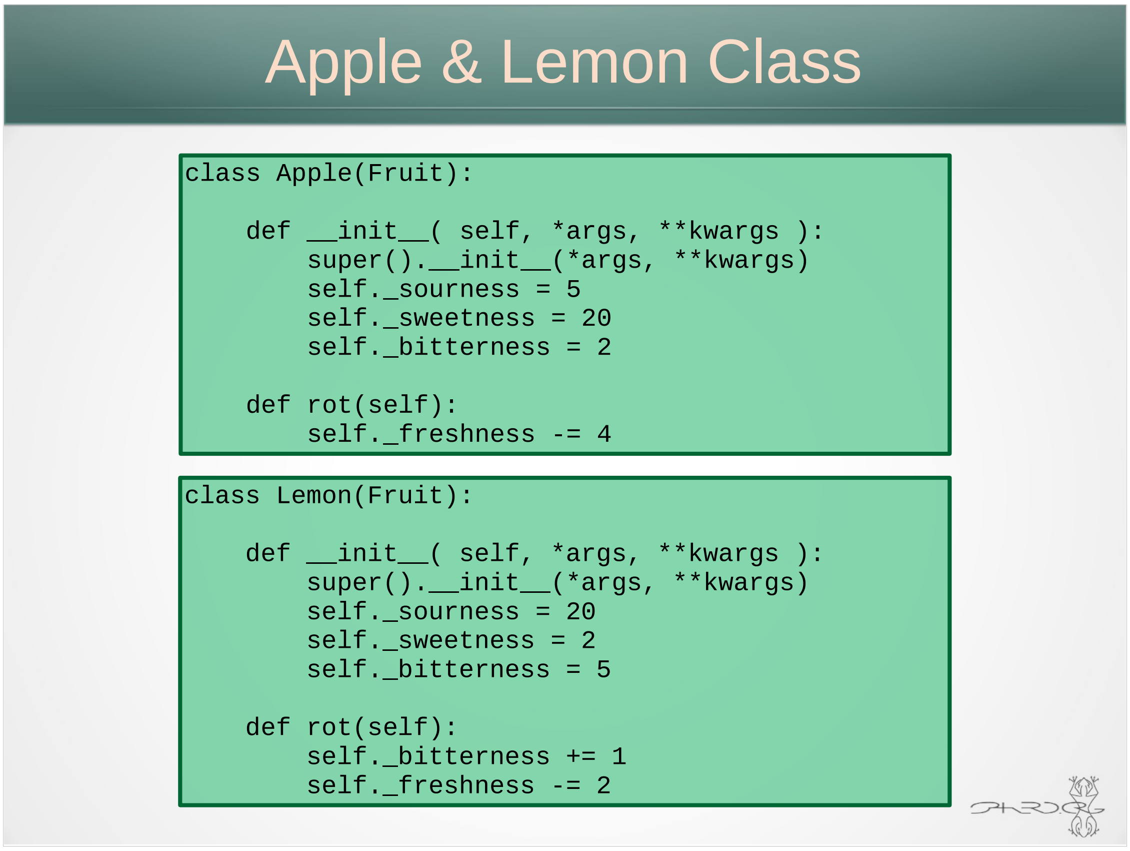


Apple Class: DIY0

```
class Apple(Fruit):

    def __init__( self, *args, **kwargs ):
        super().__init__(*args, **kwargs)
        self._sourness = 5
        self._sweetness = 20
        self._bitterness = 2

    def rot(self):
        self._freshness -= 4
```

Maak zelf een Lemon class

Zorg dat python geen syntax error geeft



Apple & Lemon Class

```
class Apple(Fruit):

    def __init__( self, *args, **kwargs ):
        super().__init__(*args, **kwargs)
        self._sourness = 5
        self._sweetness = 20
        self._bitterness = 2

    def rot(self):
        self._freshness -= 4
```

```
class Lemon(Fruit):

    def __init__( self, *args, **kwargs ):
        super().__init__(*args, **kwargs)
        self._sourness = 20
        self._sweetness = 2
        self._bitterness = 5

    def rot(self):
        self._bitterness += 1
        self._freshness -= 2
```

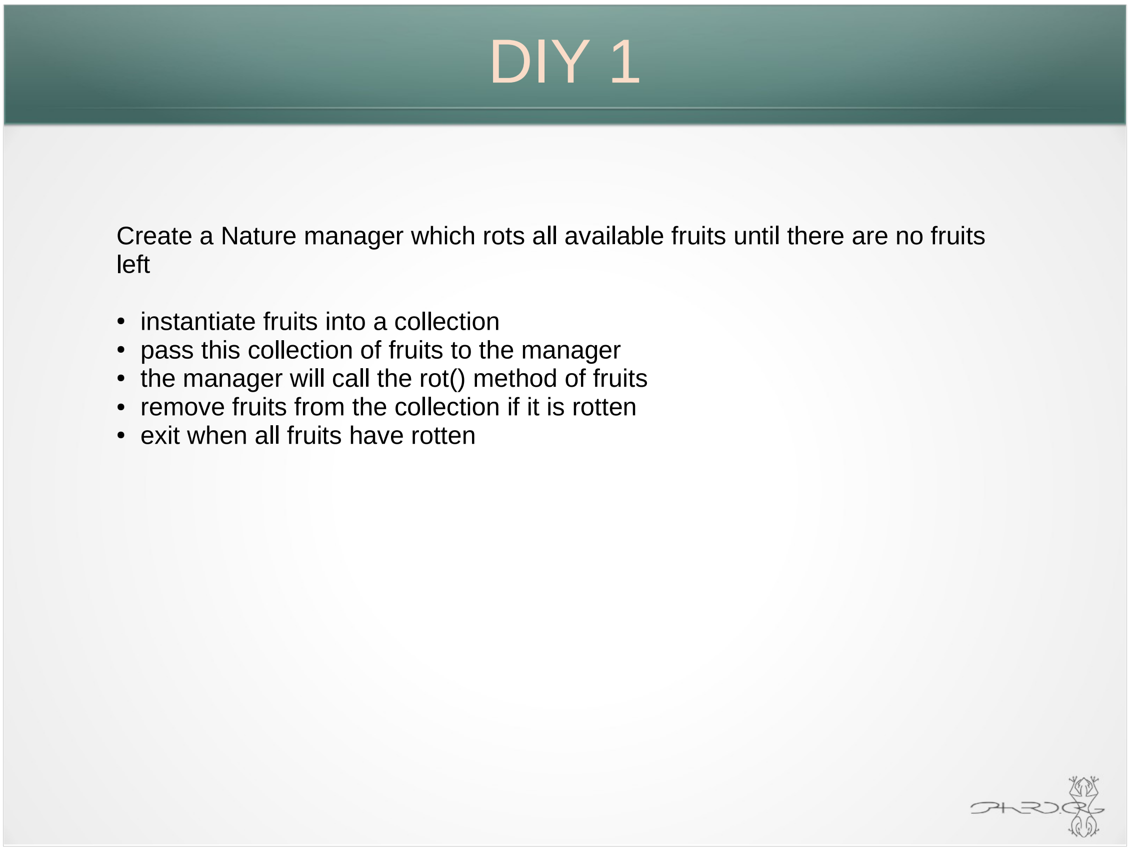
super() is anders in Py2

Instantiation

Class instances

```
a = Apple(20, 10)
l = Lemon(vit=30, weight=10)
l2 = Lemon(30, weight=11)
l3 = Lemon(weight=11, 30)
```

```
l3 = Lemon(weight=11, vit=30)
SyntaxError: non-keyword arg after keyword arg
```

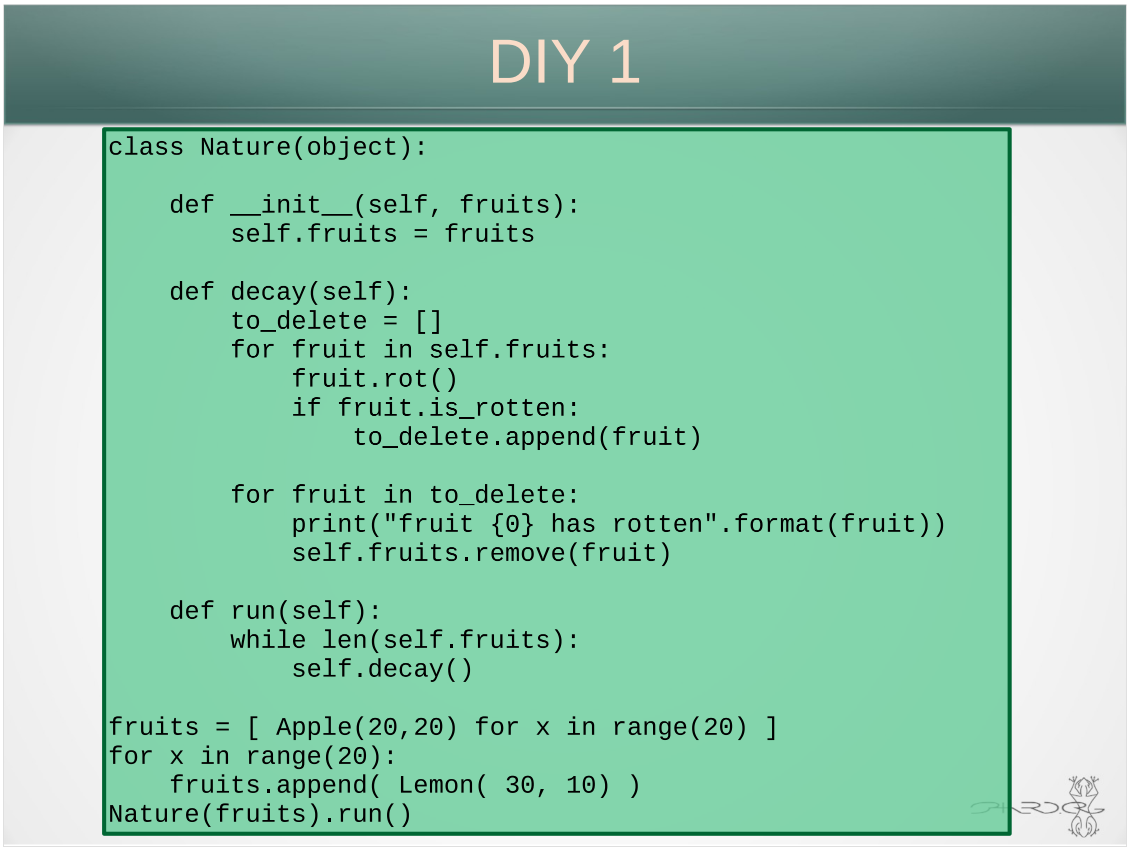


DIY 1

Create a Nature manager which rots all available fruits until there are no fruits left

- instantiate fruits into a collection
- pass this collection of fruits to the manager
- the manager will call the rot() method of fruits
- remove fruits from the collection if it is rotten
- exit when all fruits have rotten

Maak een Nature manager



DIY 1

```
class Nature(object):

    def __init__(self, fruits):
        self.fruits = fruits

    def decay(self):
        to_delete = []
        for fruit in self.fruits:
            fruit.rot()
            if fruit.is_rotten:
                to_delete.append(fruit)

        for fruit in to_delete:
            print("fruit {0} has rotten".format(fruit))
            self.fruits.remove(fruit)

    def run(self):
        while len(self.fruits):
            self.decay()

fruits = [ Apple(20,20) for x in range(20) ]
for x in range(20):
    fruits.append( Lemon( 30, 10) )
Nature(fruits).run()
```

Run DIY0 met print statements, is een bende
Toon DIY1+ met repr methoden

Standard methods

Syntax	Python calls
`x = MyClass()`	`__init__`
`del x`	`__del__`
`for i in x:`	`__iter__ / __next__`
`print(x)`	`__repr__`
`x()`	`__call__`
`len(x)`	`__len__`
`y in x:`	`__contains__`
`x + y`	`__add__`
`x - y`	`__sub__`
`x * y`	`__mul__`
`x == y`	`__eq__`
`x != y`	`__ne__`
`x > y`	`__lt__`
`x >= y`	`__le__`
`if x:`	`__bool__`

https://docs.python.org/3/reference/datamodel.html#basic-customization

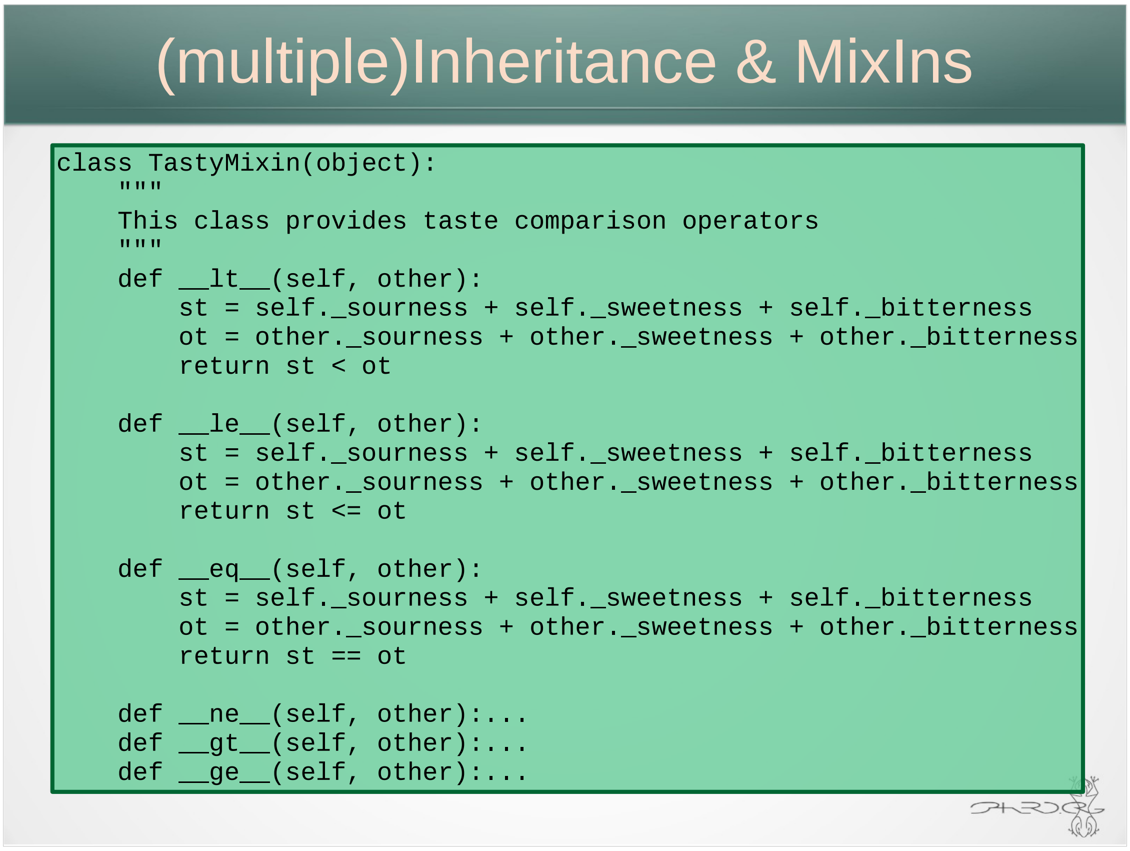
DIY 2

Add methods to determine the fruits 'tastiness'.

Make sure you can compare the fruits taste or mix them

For example what taste do we get if we sum an Apple and a Lemon

```
a = Apple()
l = Lemon()
a + l
```



(multiple)Inheritance & MixIns

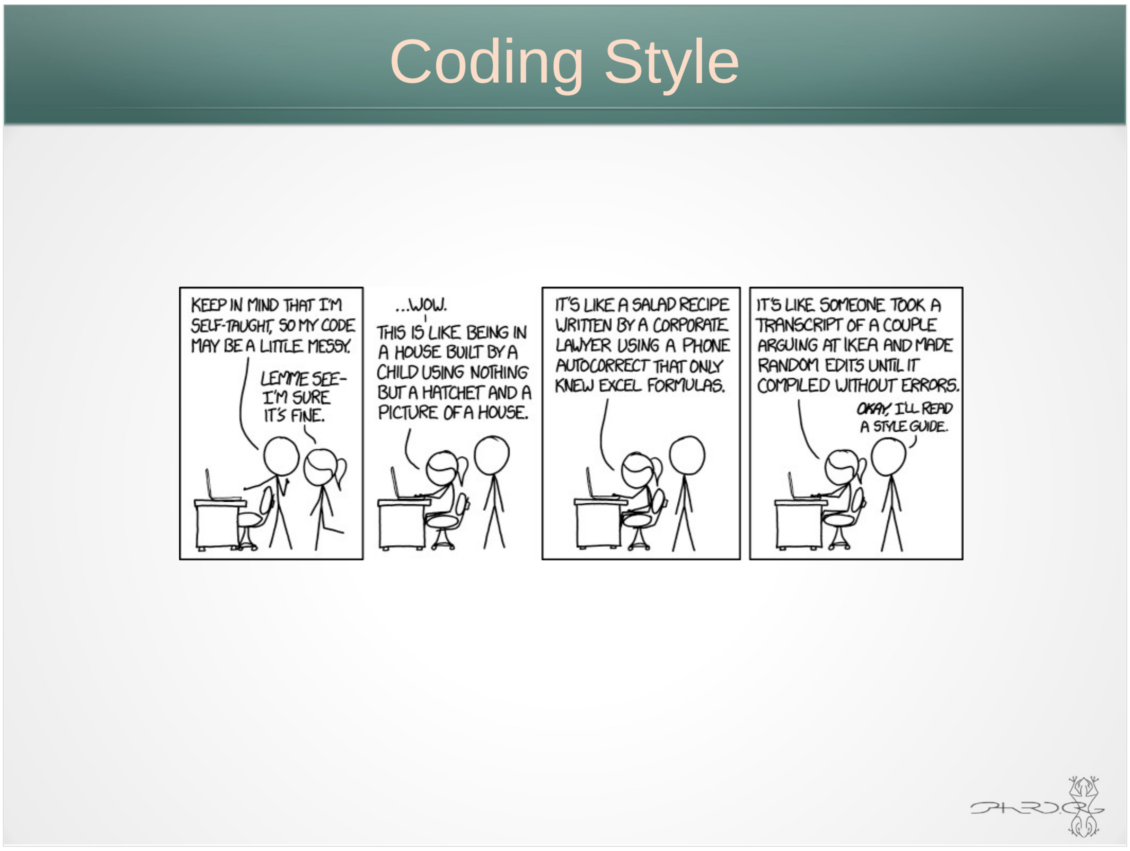
```
class TastyMixin(object):
    """
    This class provides taste comparison operators
    """
    def __lt__(self, other):
        st = self._sourness + self._sweetness + self._bitterness
        ot = other._sourness + other._sweetness + other._bitterness
        return st < ot

    def __le__(self, other):
        st = self._sourness + self._sweetness + self._bitterness
        ot = other._sourness + other._sweetness + other._bitterness
        return st <= ot

    def __eq__(self, other):
        st = self._sourness + self._sweetness + self._bitterness
        ot = other._sourness + other._sweetness + other._bitterness
        return st == ot

    def __ne__(self, other):...
    def __gt__(self, other):...
    def __ge__(self, other):...
```

Een oplossing is om een MixIn class te gebruiken

- voorkom multiple inheritance (circular dependencies & diamond deps etc)

Als je er nog niet aan toe bent: AVOID!

class Fruit(TastyMixin, object):



Coding Style

KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE— I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

Koffie? of na coding style?

Coding Style

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

PEP8: Python Style Guide

- Indentation: use 4 spaces (Spaces are preferred!)
- Align with opening delimiter:

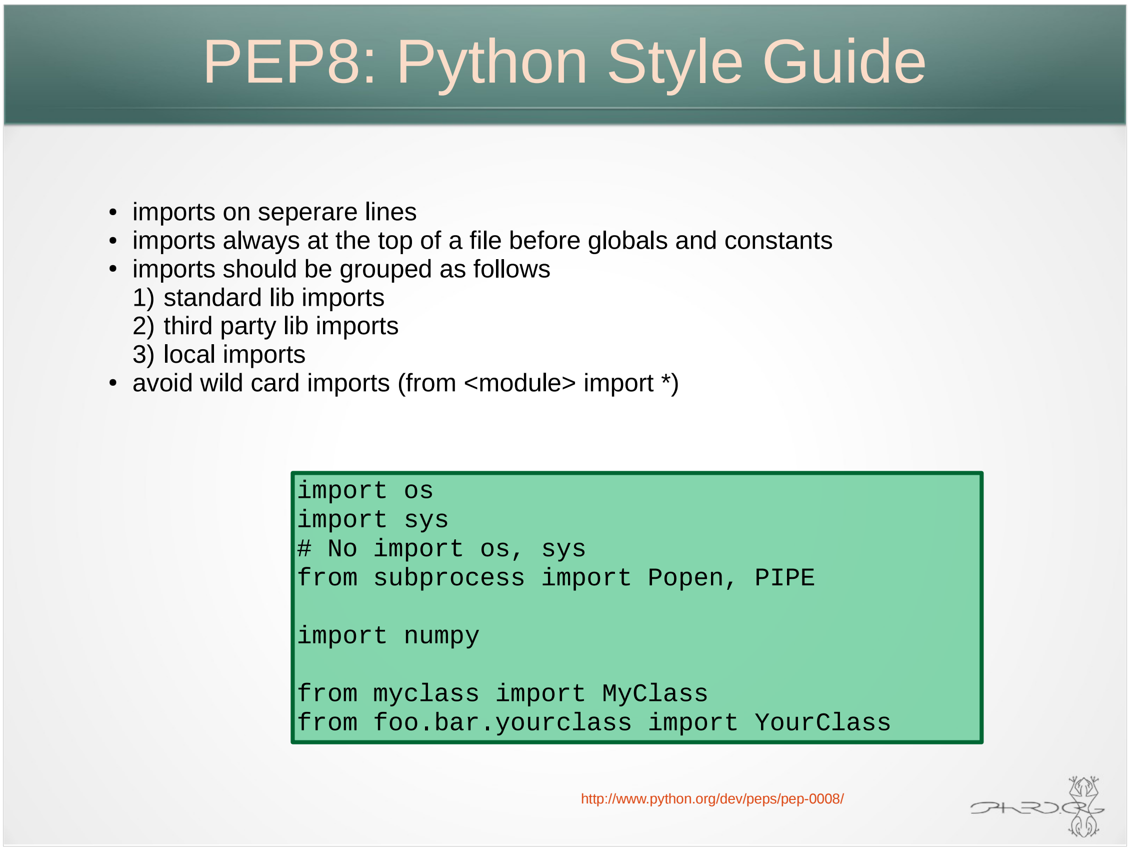
```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

- multi-line constructs line up

```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]
```

- Limit all lines to a maximum of 79 characters.
- Surround top-level function and class definitions with two blank lines.
- Method definitions inside a class are surrounded by a single blank line.
- Always use UTF-8 for source files (Python3)
- For core source only use ASCII characters

http://www.python.org/dev/peps/pep-0008/

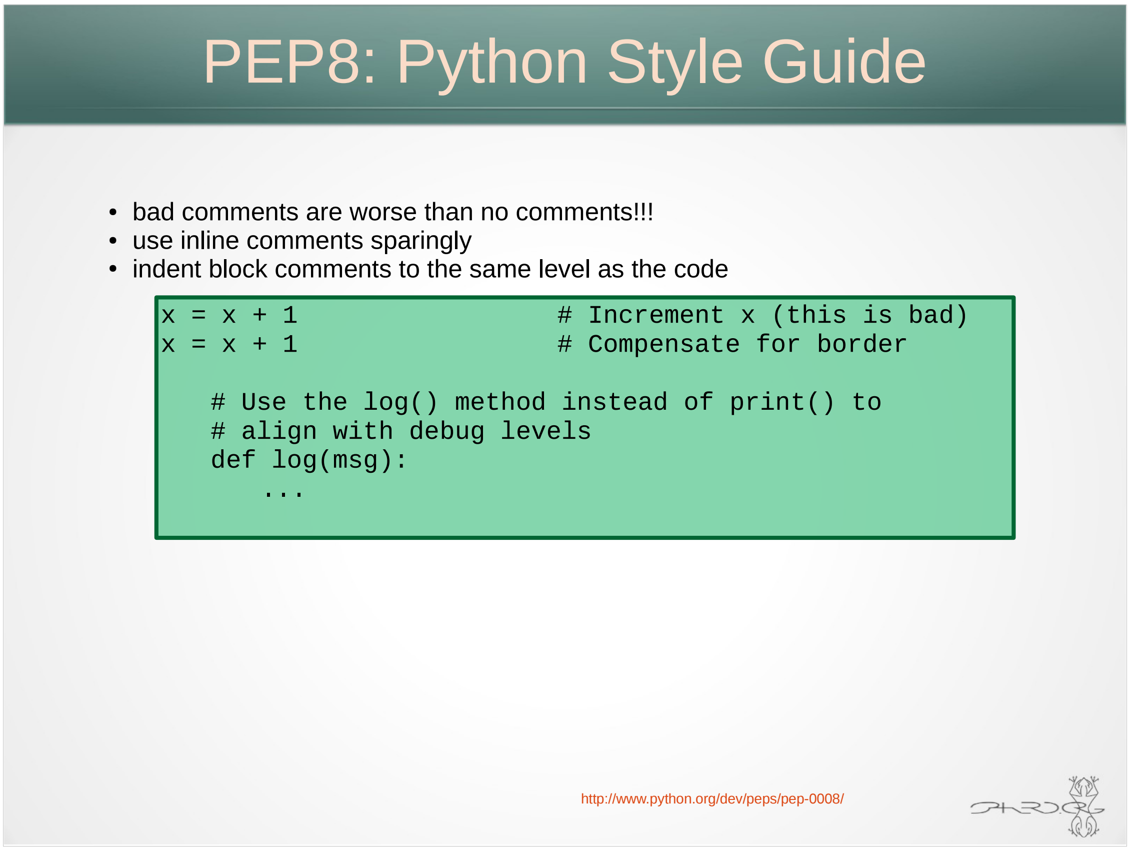


PEP8: Python Style Guide

- imports on seperare lines
- imports always at the top of a file before globals and constants
- imports should be grouped as follows
  1) standard lib imports
  2) third party lib imports
  3) local imports
- avoid wild card imports (from <module> import *)

```
import os
import sys
# No import os, sys
from subprocess import Popen, PIPE

import numpy

from myclass import MyClass
from foo.bar.yourclass import YourClass
```

http://www.python.org/dev/peps/pep-0008/



PEP8: Python Style Guide

- bad comments are worse than no comments!!!
- use inline comments sparingly
- indent block comments to the same level as the code

```
x = x + 1                 # Increment x (this is bad)
x = x + 1                 # Compensate for border

   # Use the log() method instead of print() to
   # align with debug levels
   def log(msg):
      ...
```

http://www.python.org/dev/peps/pep-0008/

PEP8: Python Style Guide

- modules should have all-lowercase names
- Class names should use CapWords
- Exceptions should use the Error suffix
- Function/method names should be lowercase with words seperated by _
- Use one leading underscore only for non-public methods and instance variables.
- Contstants all capital letters with underscores separating words
-

```
import foo.bar

BLA = 0

class CapWord(object):
    """
    Write docstrings for all public classes
    """

    _internal_var = 3

    def do_bla(self):
        if False:
            raise CapWordError(“d'oh”)
```

http://www.python.org/dev/peps/pep-0008/

DIY3

Install pep8

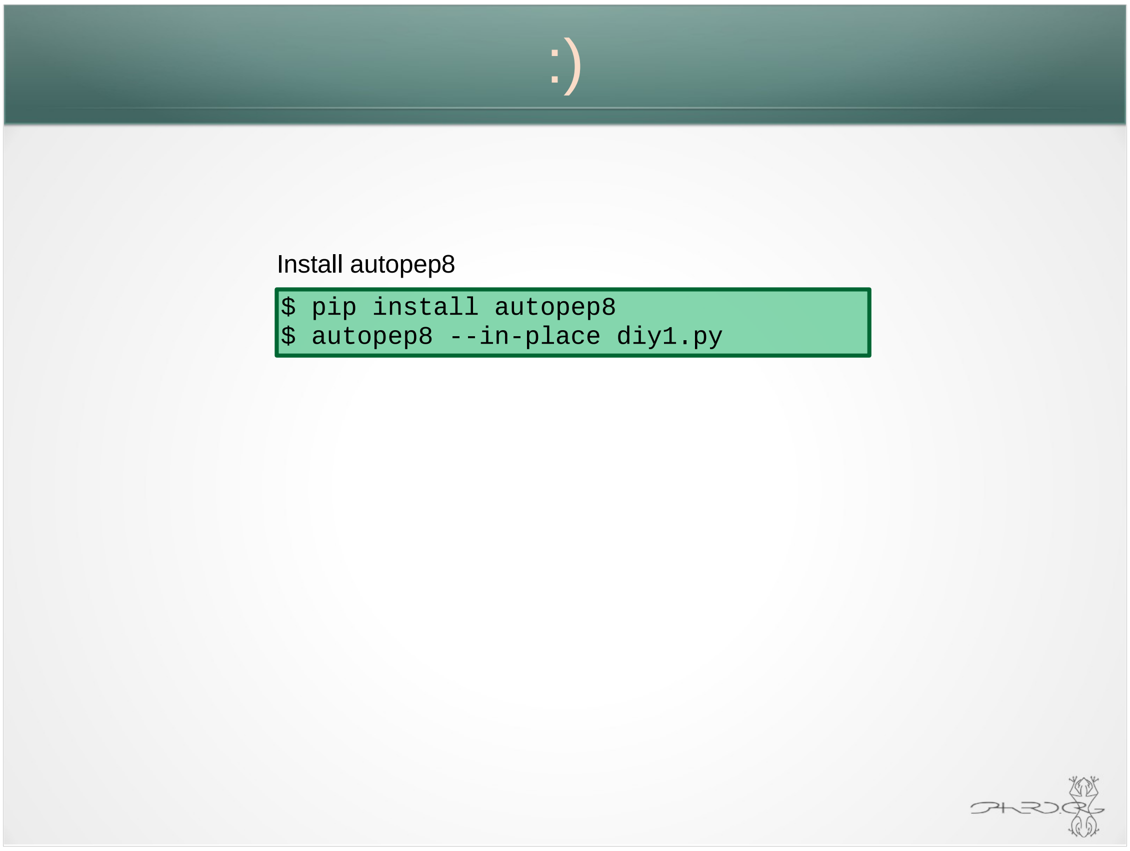
```
$ pip install pep8
```

Or through the distro package management

```
$ sudo apt-get install pep8
```

Run pep8 on your .py file containing the previous classes until no more errors

```
$ pep8 diy1.py
diy0.py:13:1: W293 blank line contains whitespace
diy0.py:22:1: W293 blank line contains whitespace
diy0.py:24:1: W293 blank line contains whitespace
diy0.py:25:18: E201 whitespace after '('
diy0.py:25:40: E202 whitespace before ')'
```



:)

Install autopep8

```
$ pip install autopep8
$ autopep8 --in-place diy1.py
```

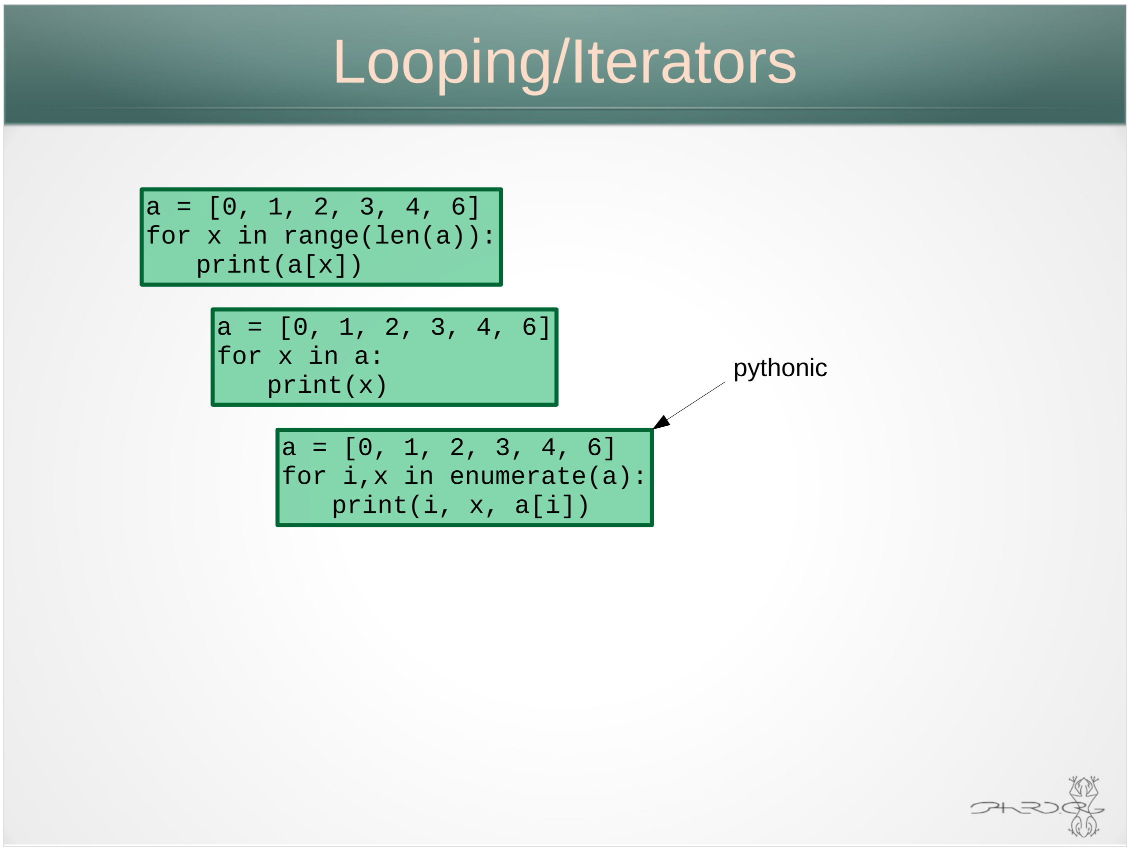
koffie

Pythonic Thinking

> all your pseudo code are belong to us

Raw Strings

```
>>> "I'm a string,\nm'kay"
"I'm a string,\nm'kay"

>>> r"I'm a string,\nm'kay"
"I'm a string,\\nm'kay"

>>> b"I'm a string,\nm'kay"
b"I'm a string,\nm'kay"

>>> u"I'm a string,\nm'kay"
"I'm a string,\nm'kay"

>>> """
... I'm a string
...
... m'kay"""
"\nI'm a string\n\nm'kay"

>>> "I'm a string,\nm'kay".upper()
"I'M A STRING,\nM'KAY"

>>> "I like the numbers {0} and {1}".format(3, 4**2)
"I like the numbers 3 and 16"
```



Looping/Iterators

```
a = [0, 1, 2, 3, 4, 6]
for x in range(len(a)):
    print(a[x])
```

```
a = [0, 1, 2, 3, 4, 6]
for x in a:
    print(x)
```

```
a = [0, 1, 2, 3, 4, 6]
for i,x in enumerate(a):
    print(i, x, a[i])
```

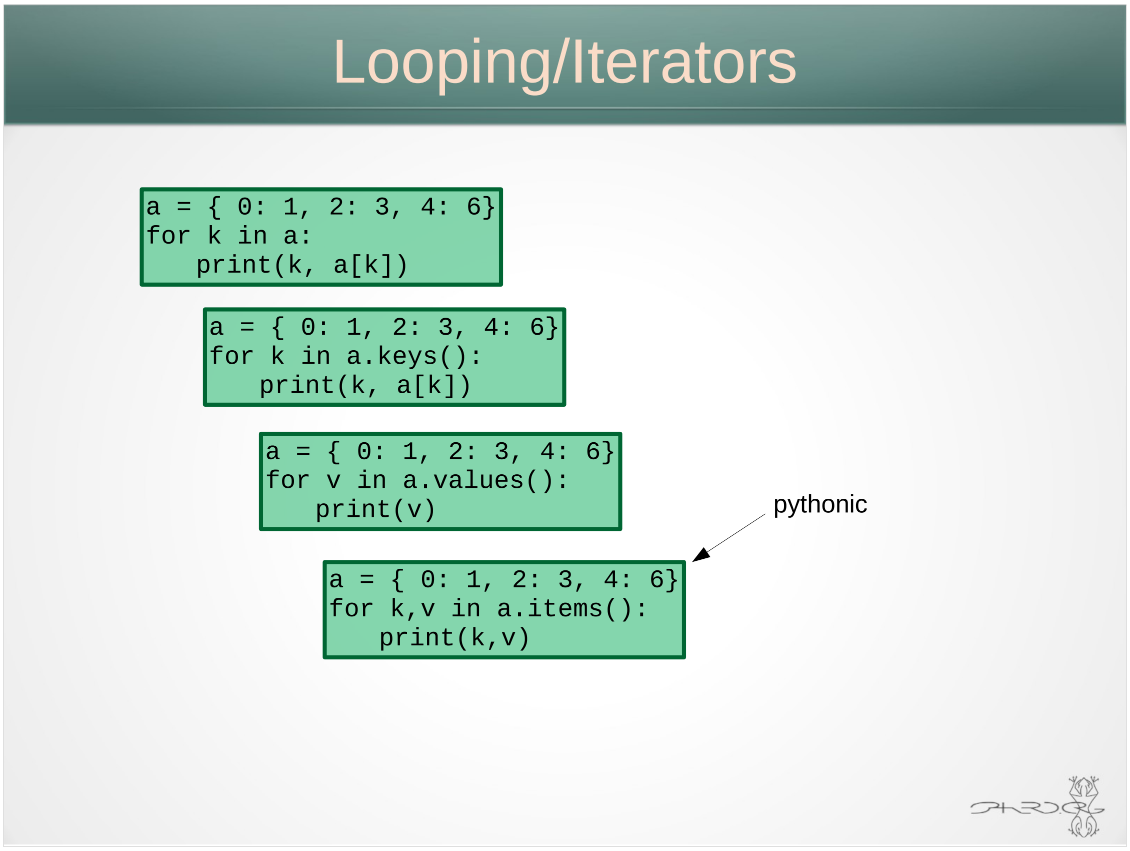
pythonic

Looping (dangerous)

```
a = [0, 1, 2, 3, 4, 6]
for x in range(len(a)):
    print(a[x])
    if x == 3:
        a.remove(a[x])
```

```
a = [0, 1, 2, 3, 4, 6]
for x in a:
    print(x)
    if x == 3:
        a.remove(x)
```

```
a = [0, 1, 2, 3, 4, 6]
for i,x in enumerate(a):
    print(i, x, a[i])
    if x == 3:
        a.remove(x)
```



Looping/Iterators

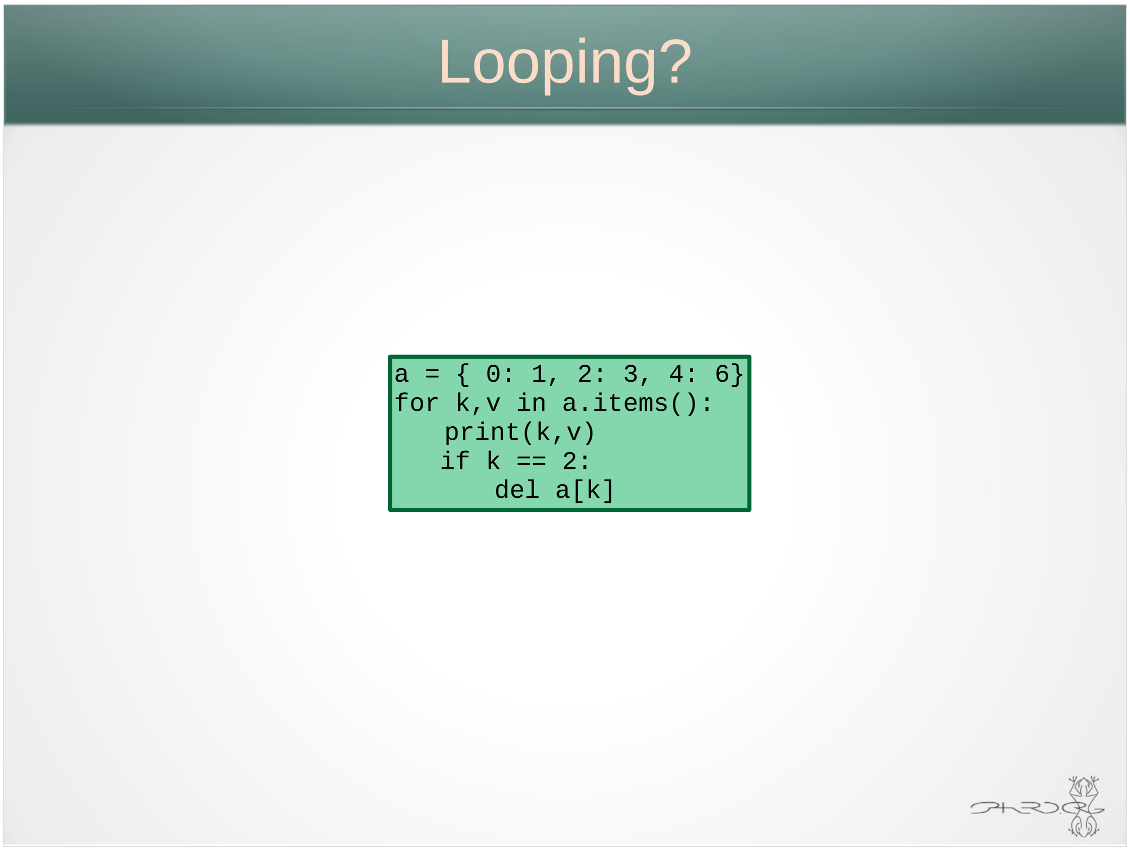
```
a = { 0: 1, 2: 3, 4: 6}
for k in a:
    print(k, a[k])
```

```
a = { 0: 1, 2: 3, 4: 6}
for k in a.keys():
    print(k, a[k])
```

```
a = { 0: 1, 2: 3, 4: 6}
for v in a.values():
    print(v)
```

pythonic

```
a = { 0: 1, 2: 3, 4: 6}
for k,v in a.items():
    print(k,v)
```



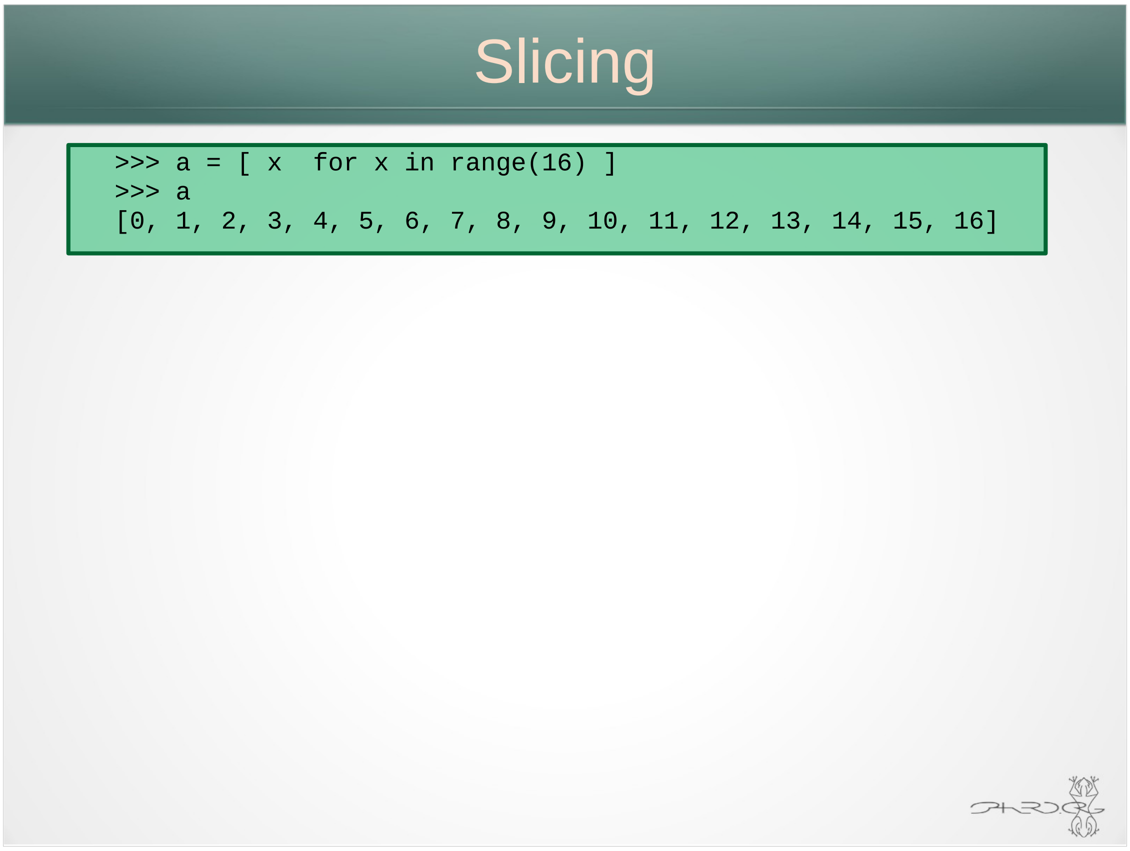
Looping?

```
a = { 0: 1, 2: 3, 4: 6}
for k,v in a.items():
    print(k,v)
    if k == 2:
        del a[k]
```

Looping/Iterators

```
q = ['name', 'quest', 'favorite color']
a = ['lancelot', 'the holy grail', 'blue']
for i, v in enumerate(q):
    print('What is your {0}? It is {1}.'.format(v, a[i]))
```

```
q = ['name', 'quest', 'favorite color']
a = ['lancelot', 'the holy grail', 'blue']
for v1, v2 in zip(q, a):
    print('What is your {0}? It is {1}.'.format(v1, v2))
```



Slicing

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

Slicing

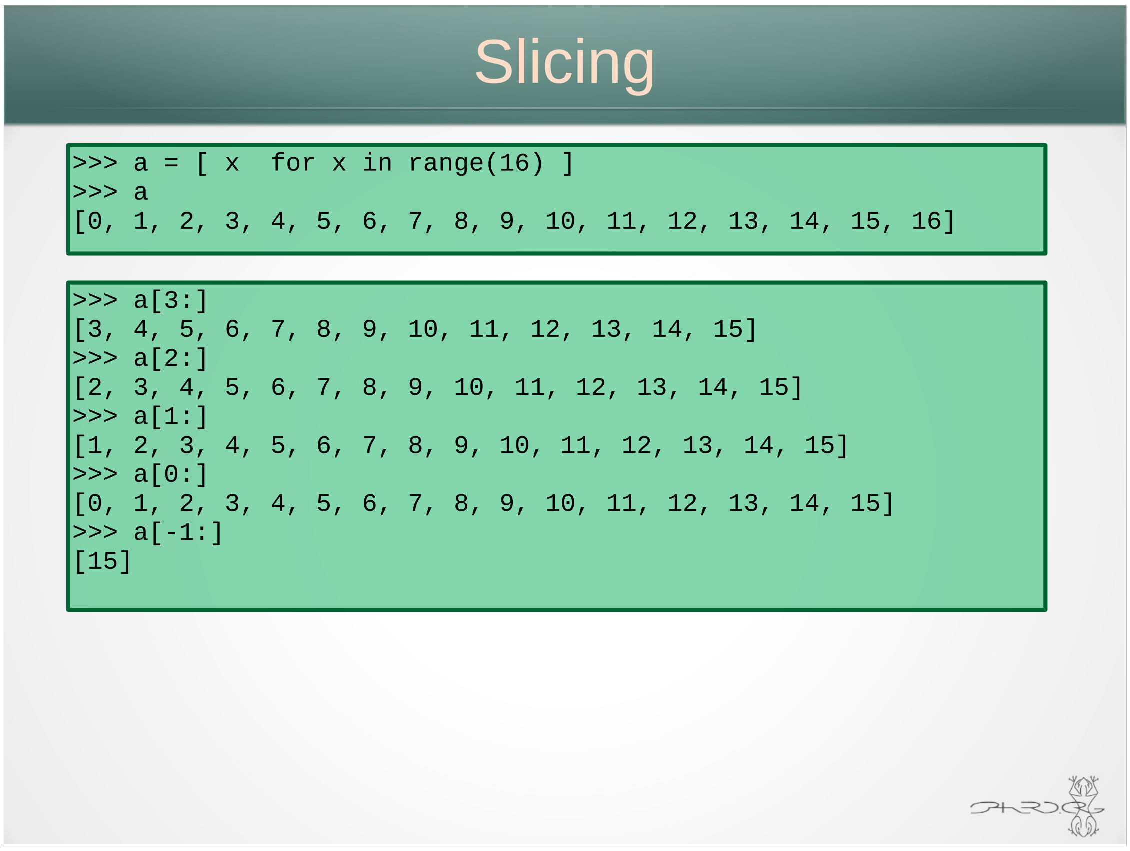
```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
>>> a[:1]
[0]
>>> a[:2]
[0, 1]
>>> a[:3]
[0, 1, 2]
```

Slicing

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
>>> a[3:]
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[2:]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[1:]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[0:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[-1:]
```

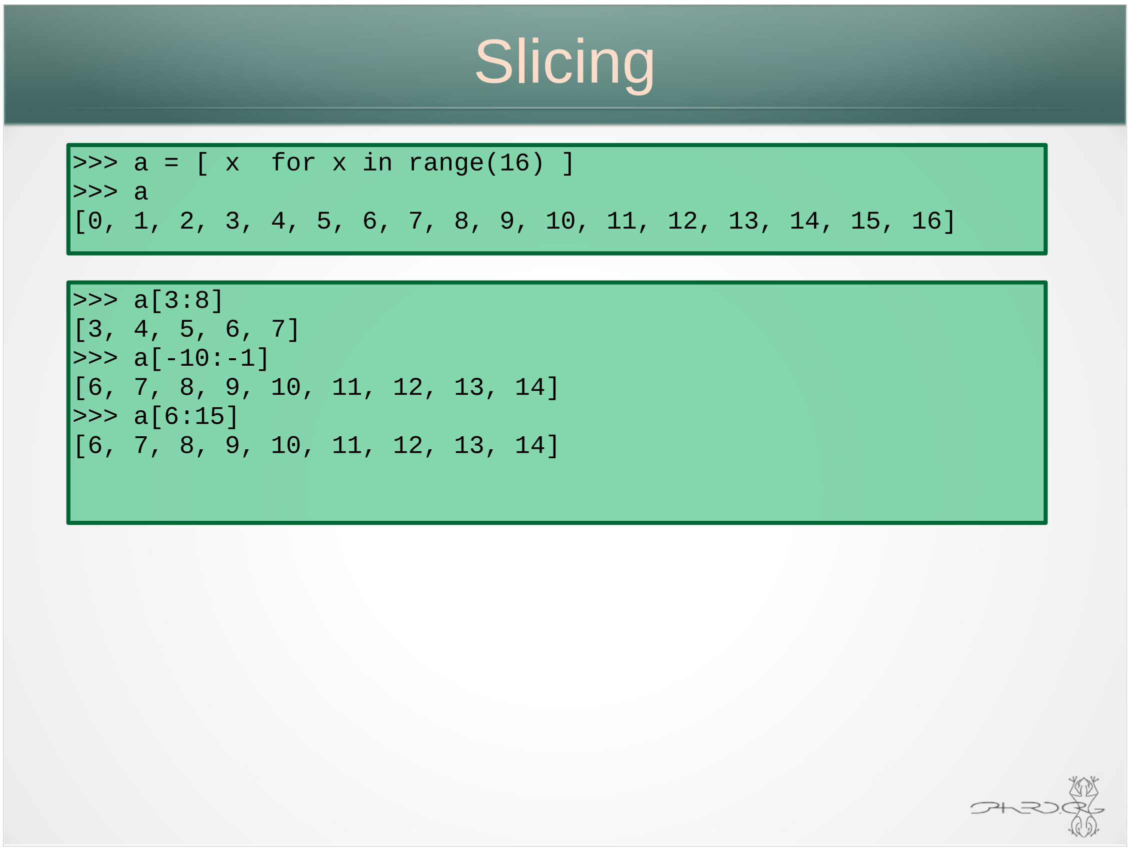


Slicing

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
>>> a[3:]
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[2:]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[1:]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[0:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> a[-1:]
[15]
```

Live in python pielen

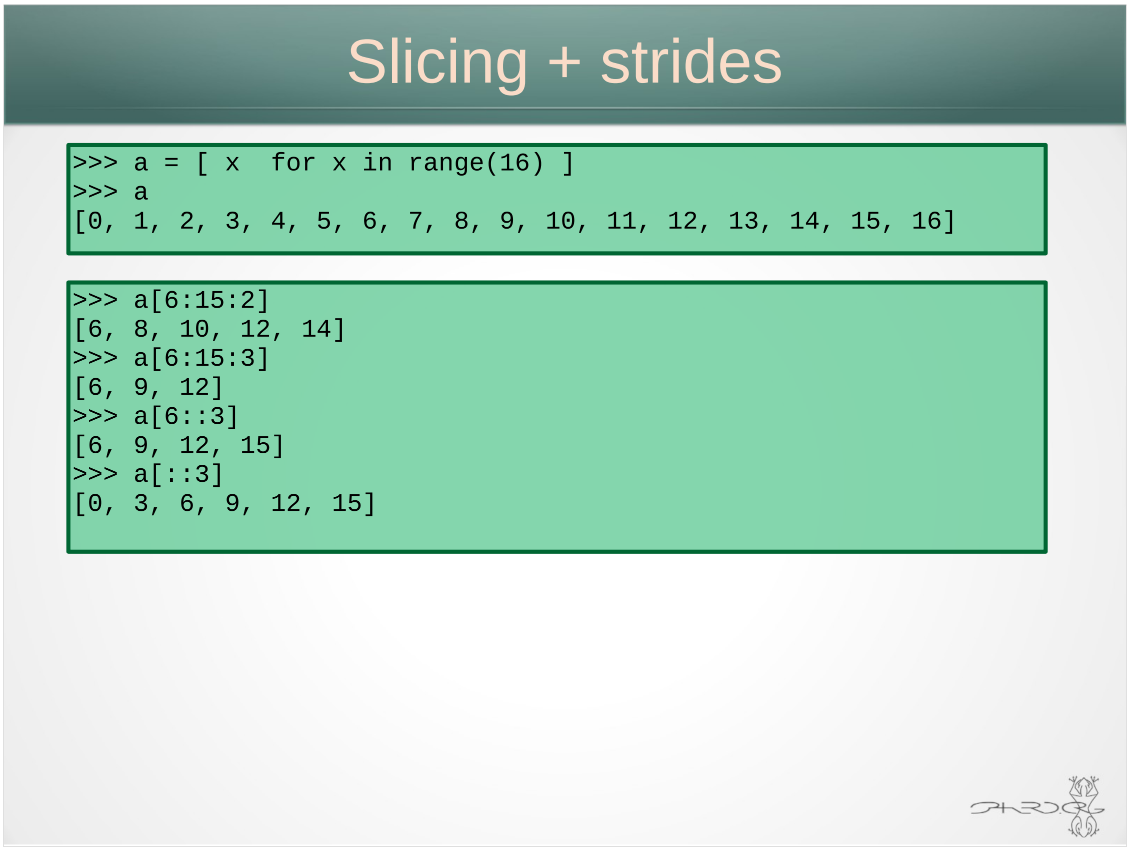


Slicing

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
>>> a[3:8]
[3, 4, 5, 6, 7]
>>> a[-10:-1]
[6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> a[6:15]
[6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Live in python pielen

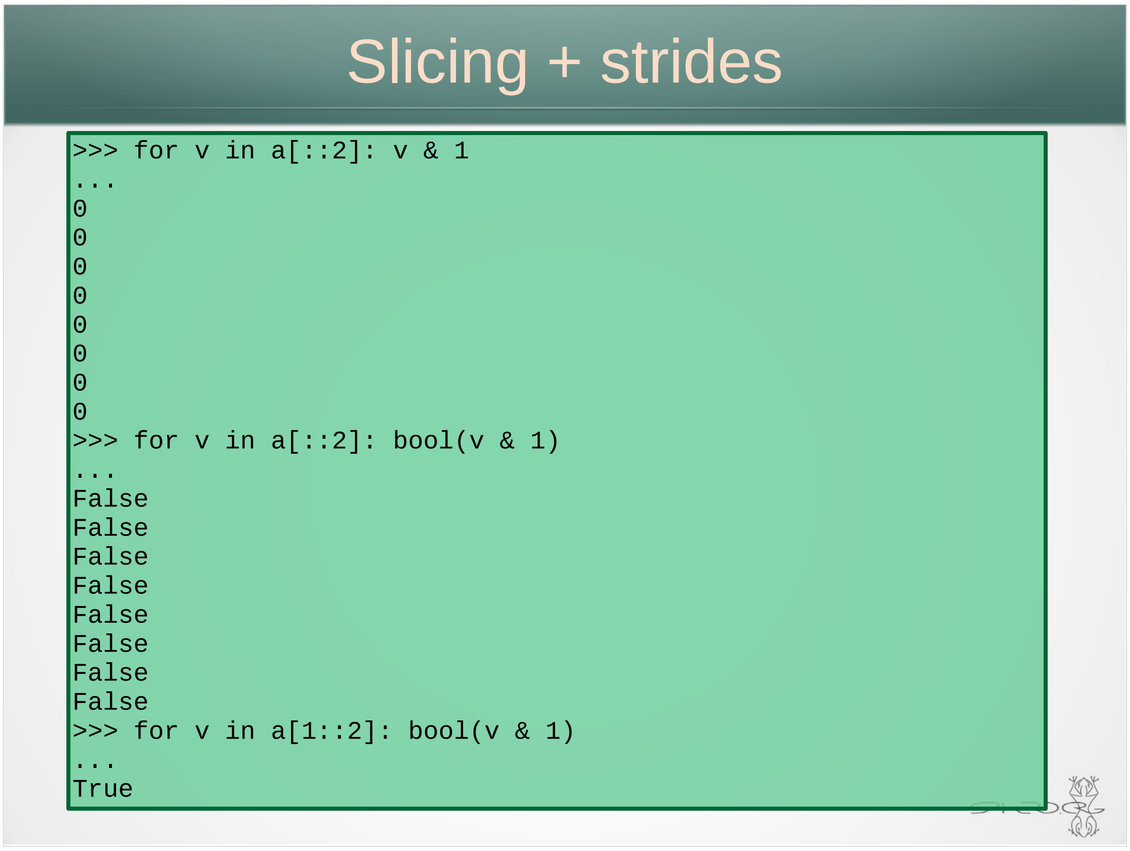


Slicing + strides

```
>>> a = [ x  for x in range(16) ]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
>>> a[6:15:2]
[6, 8, 10, 12, 14]
>>> a[6:15:3]
[6, 9, 12]
>>> a[6::3]
[6, 9, 12, 15]
>>> a[::3]
[0, 3, 6, 9, 12, 15]
```

Live in python pielen



Slicing + strides

```
>>> for v in a[::2]: v & 1
...
0
0
0
0
0
0
0
0
>>> for v in a[::2]: bool(v & 1)
...
False
False
False
False
False
False
False
False
>>> for v in a[1::2]: bool(v & 1)
...
True
```

Live in python pielen

Exceptions

```
cupcakes = 9

def yo_mama():
    global cupcakes
    cupcakes -= 1
    if cupcakes:
        return cupcakes
    # implicit return None

for x in range(9):
    print(yo_mama())
```

Exceptions

```
def yo_mama2():
    global cupcakes
    cupcakes -= 1
    if cupcakes < 3:
        return cupcakes, "We need moa"
    elif cupcakes:
        return cupcakes
    # implicit return None

cupcakes = 9
for x in range(9):
    print(yo_mama2())
```

Exceptions

```
class OutOfCupCakesError(Exception):
    """We're out of cupcakes"""

def yo_mama3():
    global cupcakes
    cupcakes -= 1
    if cupcakes < 3:
        raise OutOfCupCakesError
    elif cupcakes:
        return cupcakes
    # implicit return None

cupcakes = 9
for x in range(9):
    try:
        print(yo_mama3())
    except OutOfCupCakesError:
        print("we need moa")
```

Exceptions

```
try:
   do_something()
   do_anotherthing()
except SomeError as e:
   print(e)
else:
   print("Jay, we managed")
finally:
   print("Die peacefully")
```

Exceptions

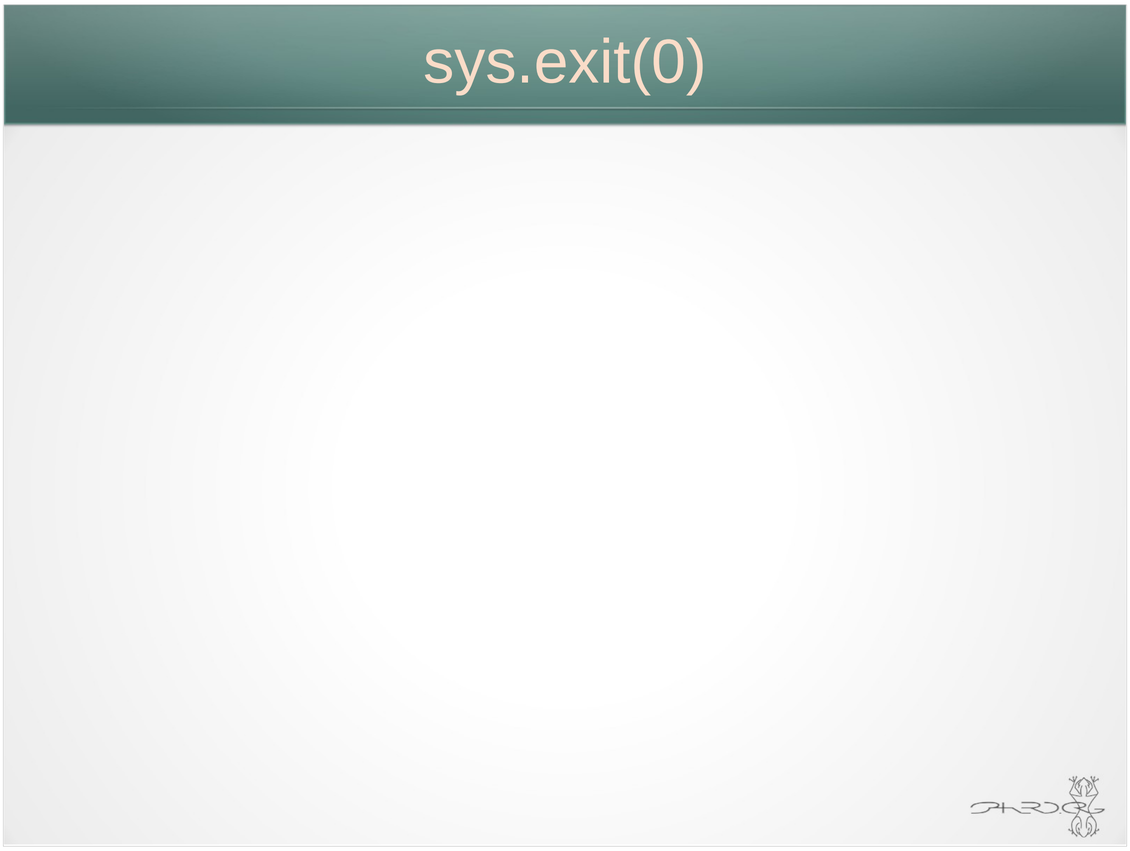
```
try:
   do_something()
   do_anotherthing()
except (SomeError, AnotherError) as e:
   print(e)
except:
   print("I give up")
else:
   print("Jay, we managed")
finally:
   print("Die peacefully")
```

DIY4

Add exceptions to the fruit classes and nature managers

make the nature fruit class iterable by taste?

or anything you picked up today



sys.exit(0)

Wat nog te doen?
Vragen?
NamedTuples
Inherit from collections
default arguments use None!!!

```
from flask import Flask
app = Flask(__main__)

@app.route(“/”)
def hello():
	return “Hello World”

if __name__ == “__main__”:
	app.run()
```