PRACTICAL LECTURE
Python and GIS scripting

Fredrik Lindberg

Urban Climate Group
Department of Earth Sciences
University of Gothenburg

Introduction to Python

A readable, dynamic, pleasant,
flexible, fast and powerful language

What is Python

- Multi-purpose (Web, GUI, Scripting, etc.)
- Object oriented
- Interpreted
- Strongly typed and Dynamically typed
- Case sensitive
- Focus on readability and productivity

Features

- Built-in libraries included (array, datetime, math, os, …)
- Everything is an Object
- Interactive Shell (through cmd)
- Strong introspection
- Cross platform
- CPython, Jython, IronPython, PyPy

Who uses Python

- Google
- Facebook
- Spotify
- Netflix
- Reddit
- NASA
- …

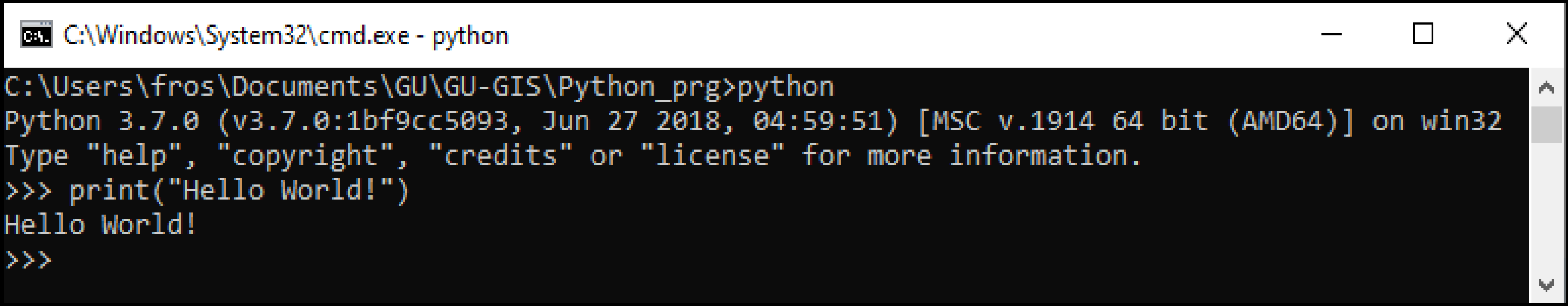
Releases

- Created in 1989 by Guido Van Rossum
- Python 1.0 released in 1994
- Python 2.0 released in 2000
- Python 3.0 released in 2008
- Python 2.18(?) is the last 2.x version (deprecated)

- Python 3.9 is the latest version

Syntax

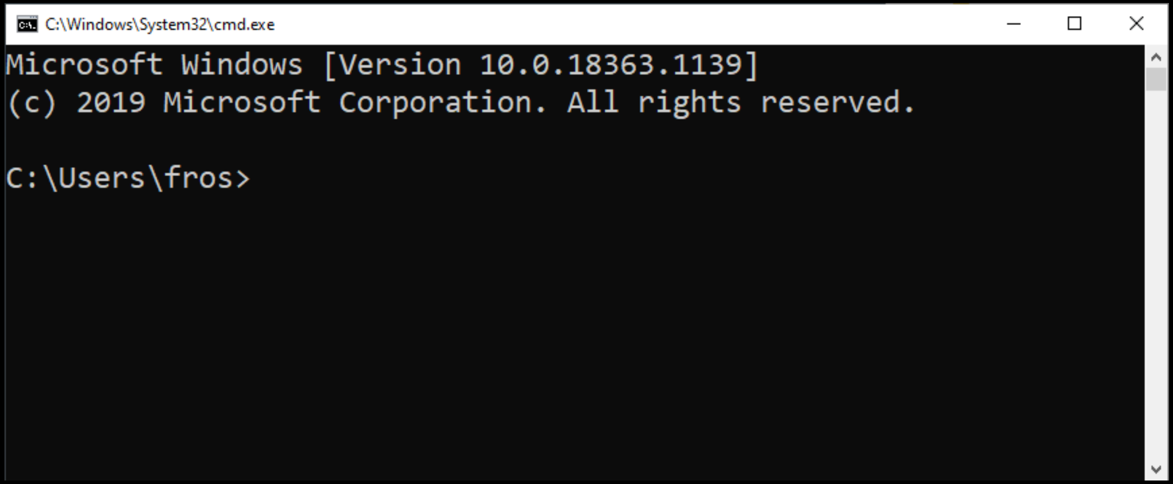
Hello World



```
C:\Users\fros\Documents\GU\GU-GIS\Python_prg>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World!")
Hello World!
>>>
```

Python on Windows

- Open a command prompt
- Type python. What happens?
- Type path



```
Microsoft Windows [Version 10.0.18363.1139]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\fros>
```

Windows dos

- Move forward in folder system cd *name*
- Auto-complete path names tab
- Content in folder dir (in Linux ls)
- Move backward in folder system cd ..
- Move back to root cd \
- Change root, e.g. D:
- Script files should end with .bat
- Add environment path path %path%;C:\OSGeo4W64\bin

More commands: e.g. http://www.computerhope.com/msdos.htm

Start Python

We can use the python installation that comes with QGIS and OSGeo by either using the *OSGeo4W Shell* OR executing our own script (win: .bat, mac: .sh).

1. Go to Github and download *QGIS3_shell.bat*
2. Edit the first row in the script to fit your *OSGeo path*
3. Open a command prompt and navigate to the folder where you store your script - execute the script
4. Type python to see if your cmd finds a python installation

Why learn programming?

- Automated processors
- Better understanding of the computer and the software used
- Don’t do the same manual work yet again
- MORE..?

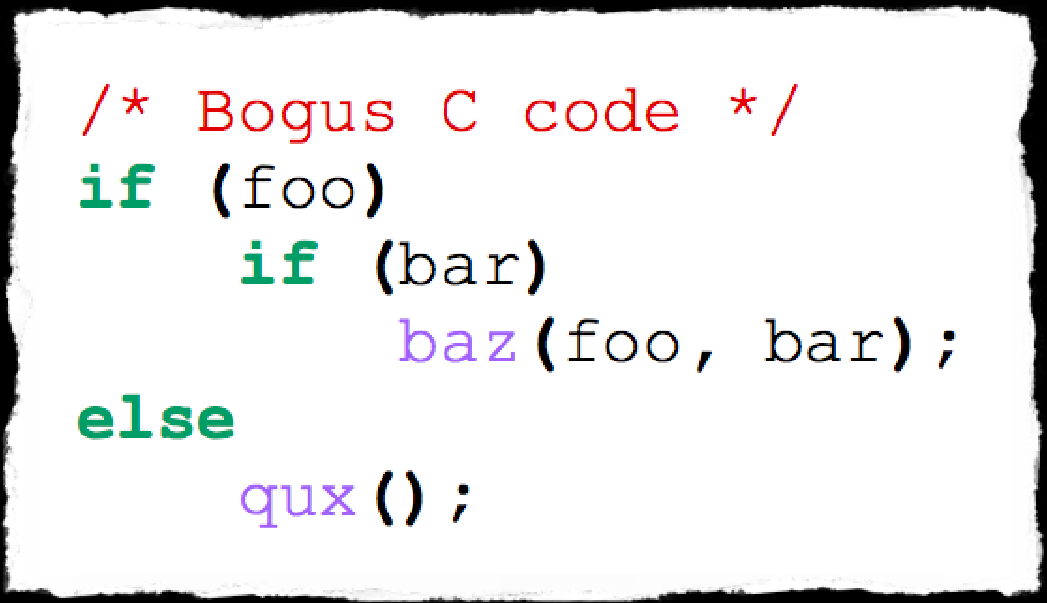
What is a loop?

- For automated processes
- Iterate over some data (features, row of numbers (vector), matrix)
- example

Indentation

- For improved readability
- Most languages don’t care about indentation
- Most humans do - we tend to group similar things together

inadequate Indentation

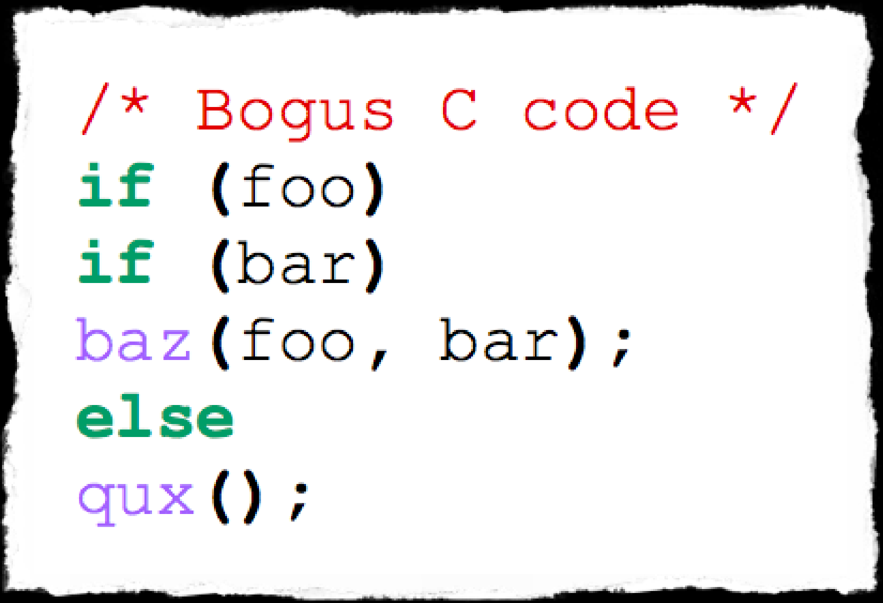


```
/* Bogus C code */
if (foo)
    if (bar)
        baz(foo, bar);
else
    qux();
```

The *else* statement (in this C code) belongs to the *2nd* if statement, even though it looks like it belongs to the first.

Python uses indentation for *readability* AND for *functionality*.

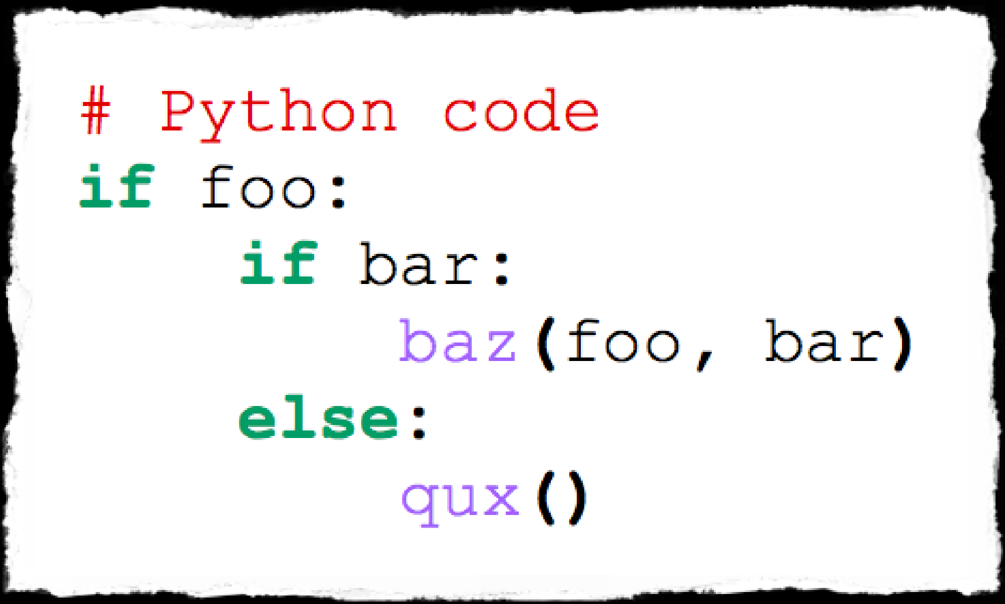
no Indentation



```
/* Bogus C code */
if (foo)
if (bar)
baz(foo, bar);
else
qux();
```

sometimes you see coding like this

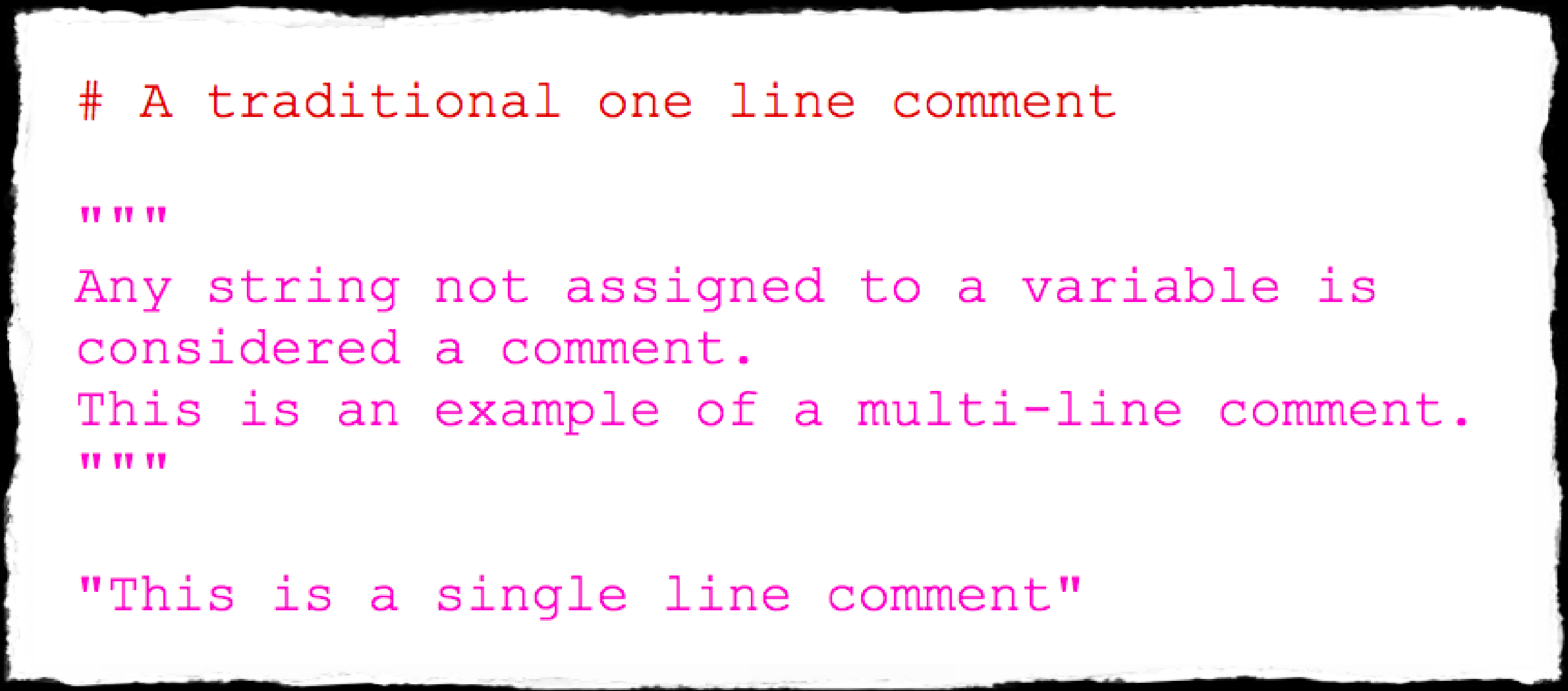
Indentation



```python
# Python code
if foo:
    if bar:
        baz(foo, bar)
    else:
        qux()
```

Python embraces indentation

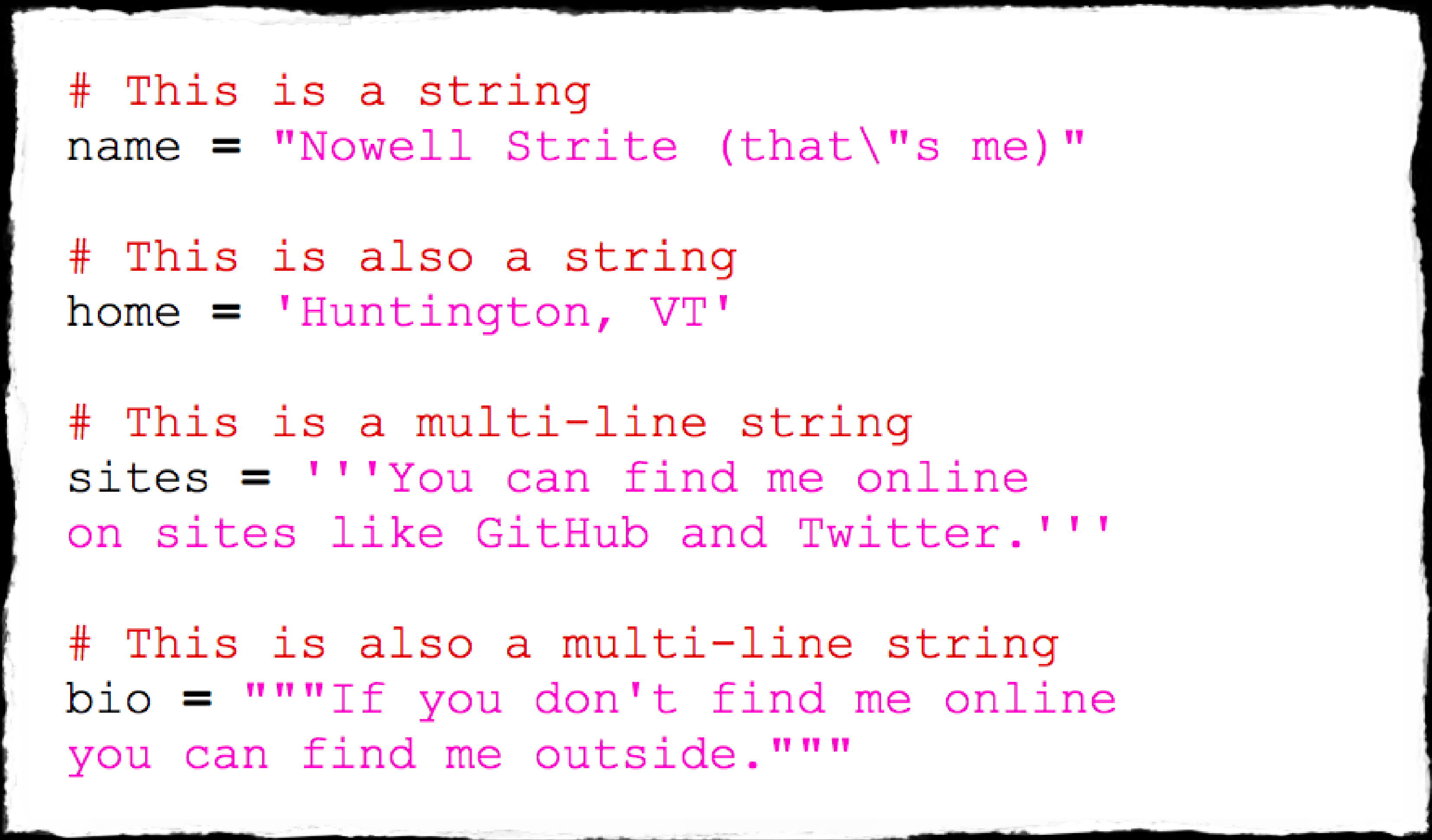
Comments



```
# A traditional one line comment

"""
Any string not assigned to a variable is
considered a comment.
This is an example of a multi-line comment.
"""

"This is a single line comment"
```

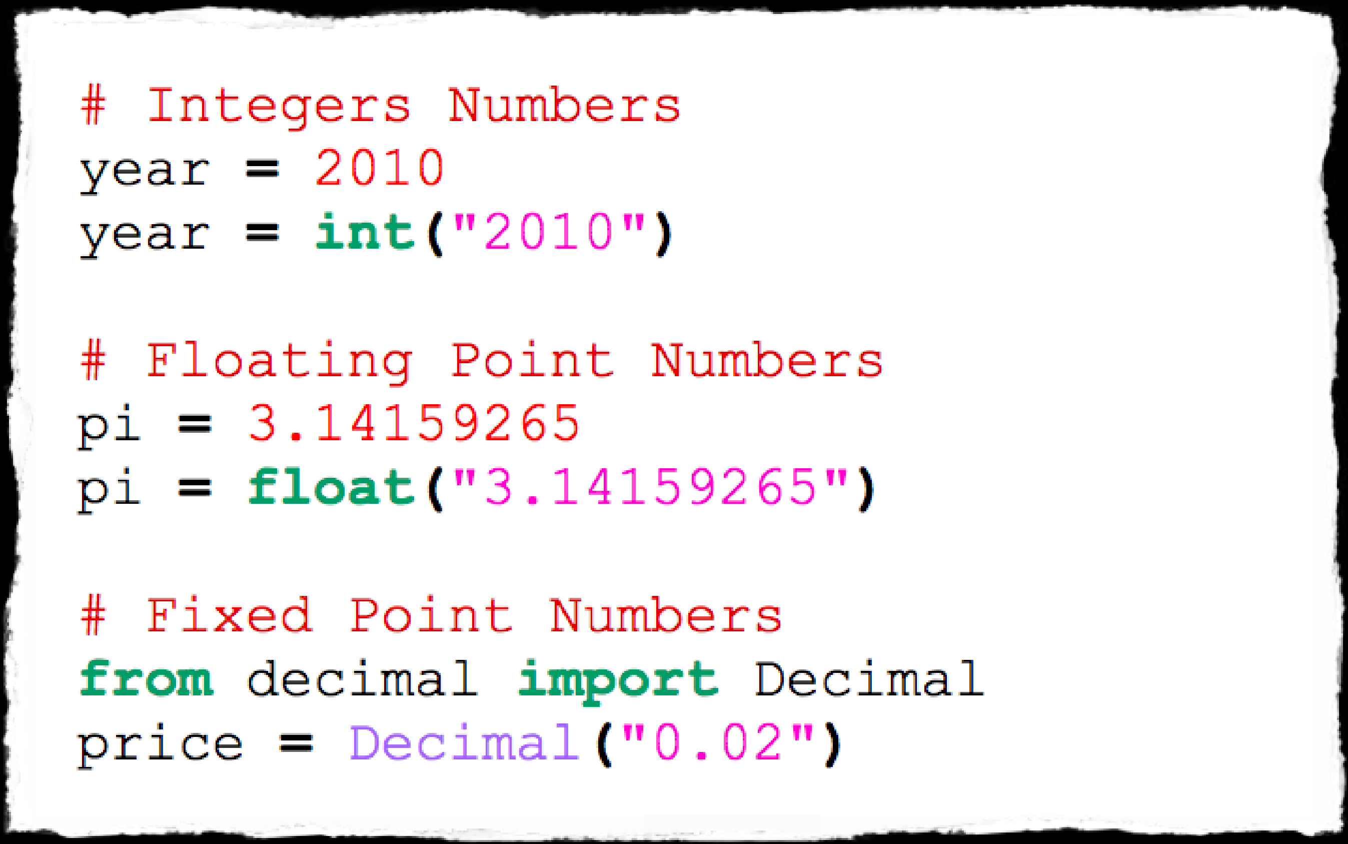
Types of variables

Strings



```
# This is a string
name = "Nowell Strite (that\"s me)"

# This is also a string
home = 'Huntington, VT'

# This is a multi-line string
sites = '''You can find me online
on sites like GitHub and Twitter.'''

# This is also a multi-line string
bio = """If you don't find me online
you can find me outside."""
```

Numbers

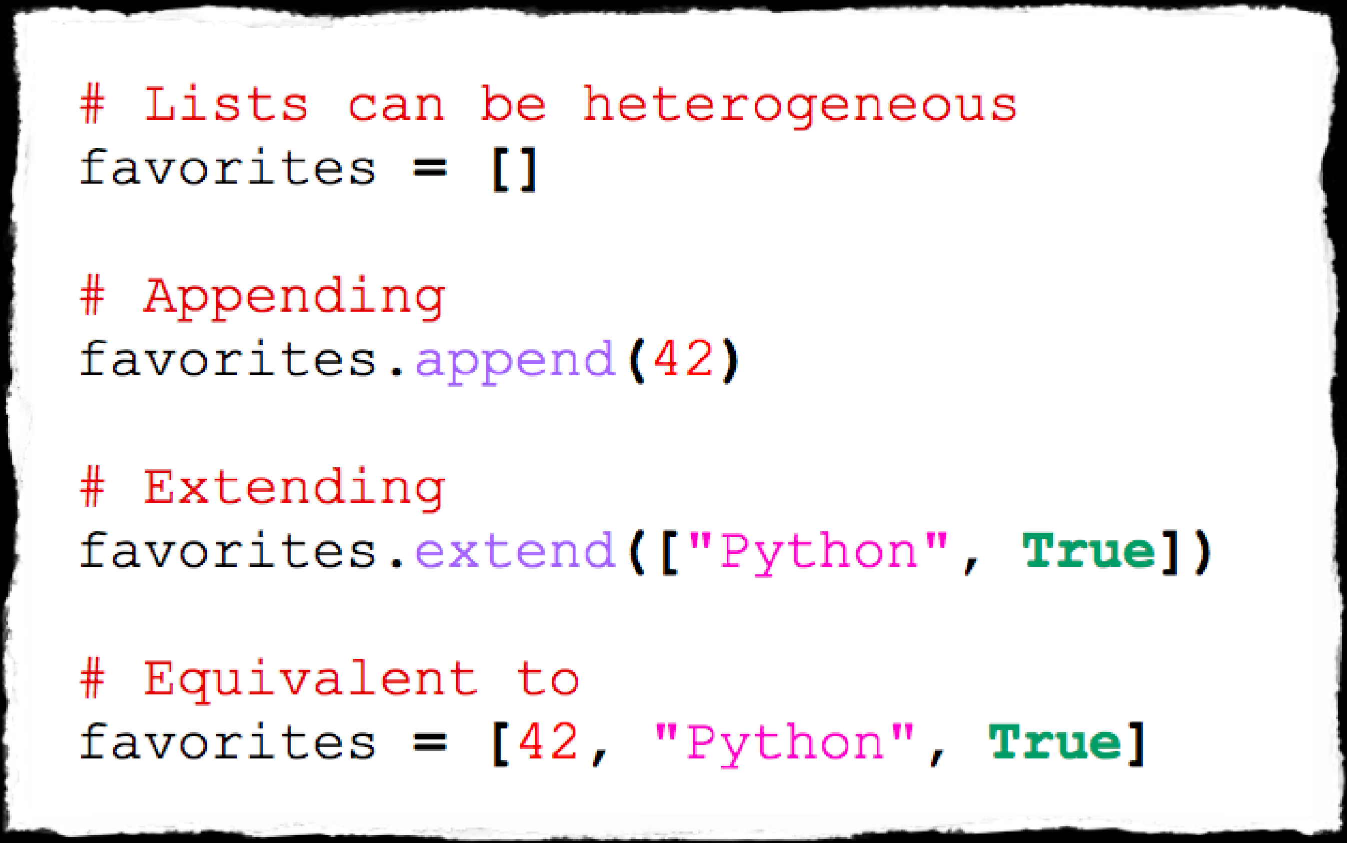


```python
# Integers Numbers
year = 2010
year = int("2010")

# Floating Point Numbers
pi = 3.14159265
pi = float("3.14159265")

# Fixed Point Numbers
from decimal import Decimal
price = Decimal("0.02")
```

Null

```
optional_data = None
```

Lists

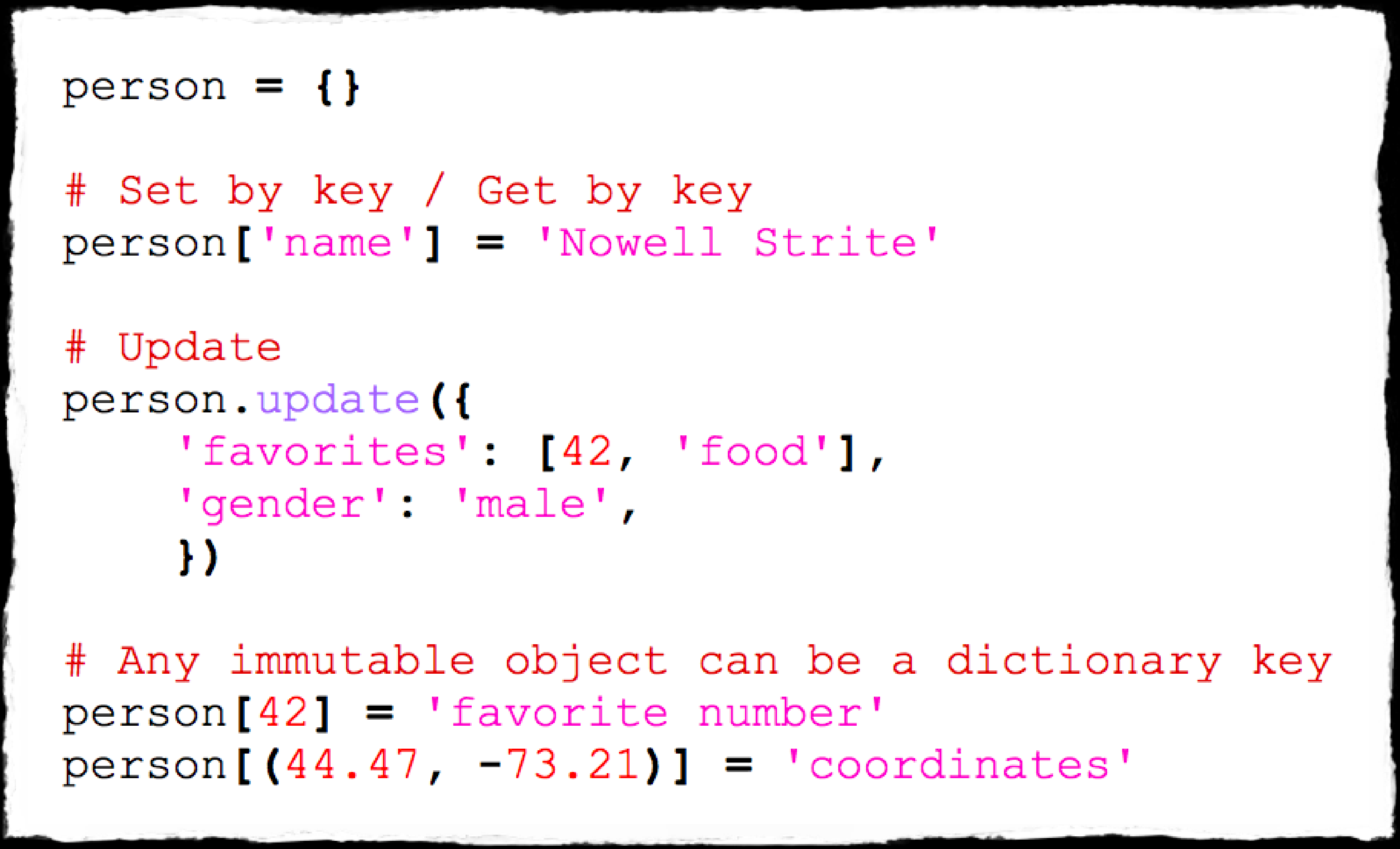


```
# Lists can be heterogeneous
favorites = []

# Appending
favorites.append(42)

# Extending
favorites.extend(["Python", True])

# Equivalent to
favorites = [42, "Python", True]
```

Lists

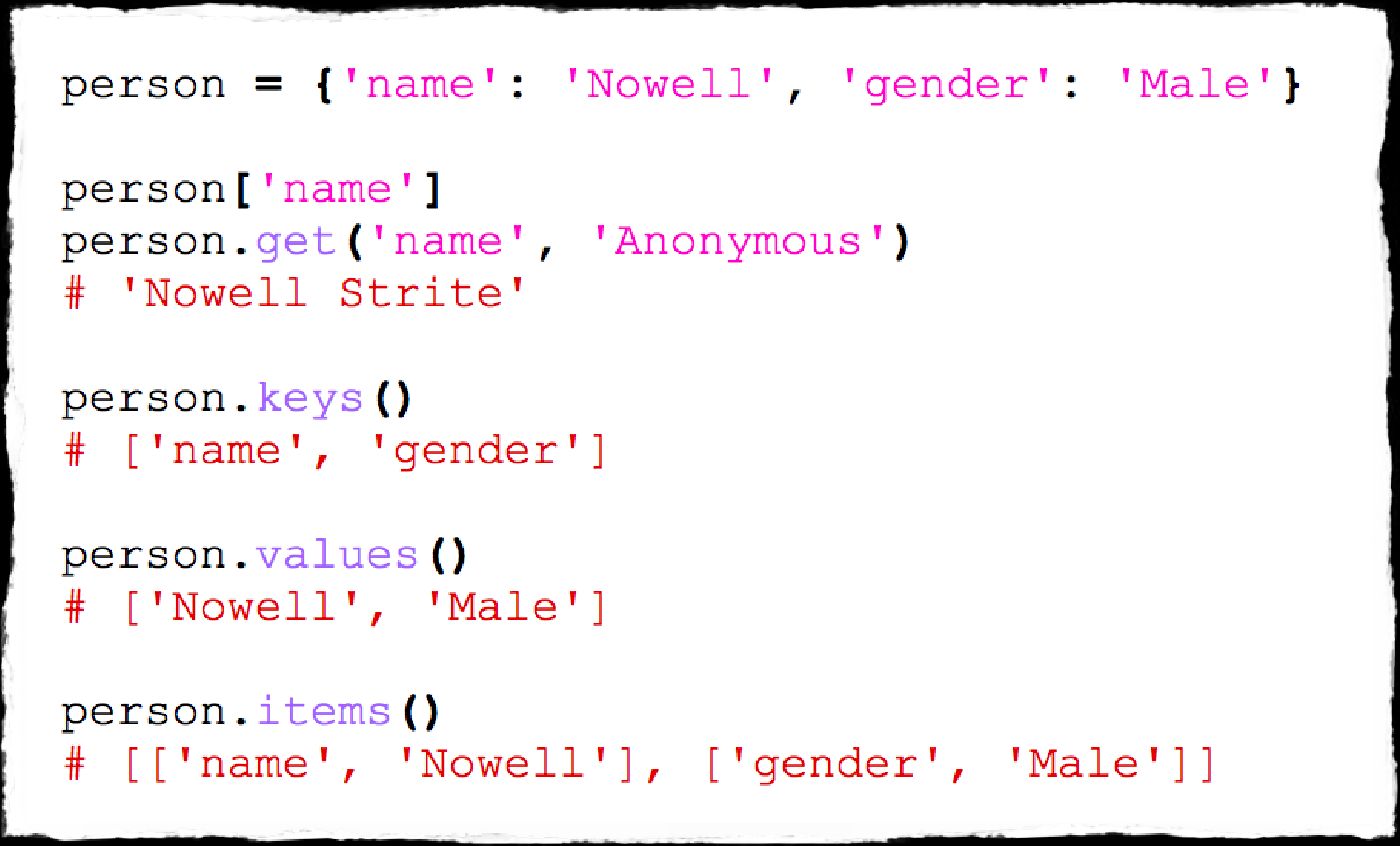
```
numbers = [1, 2, 3, 4, 5]

len(numbers)
# 5

numbers[0]
# 1

numbers[0:2]
# [1, 2]

numbers[2:]
# [3, 4, 5]
```

Dictionaries



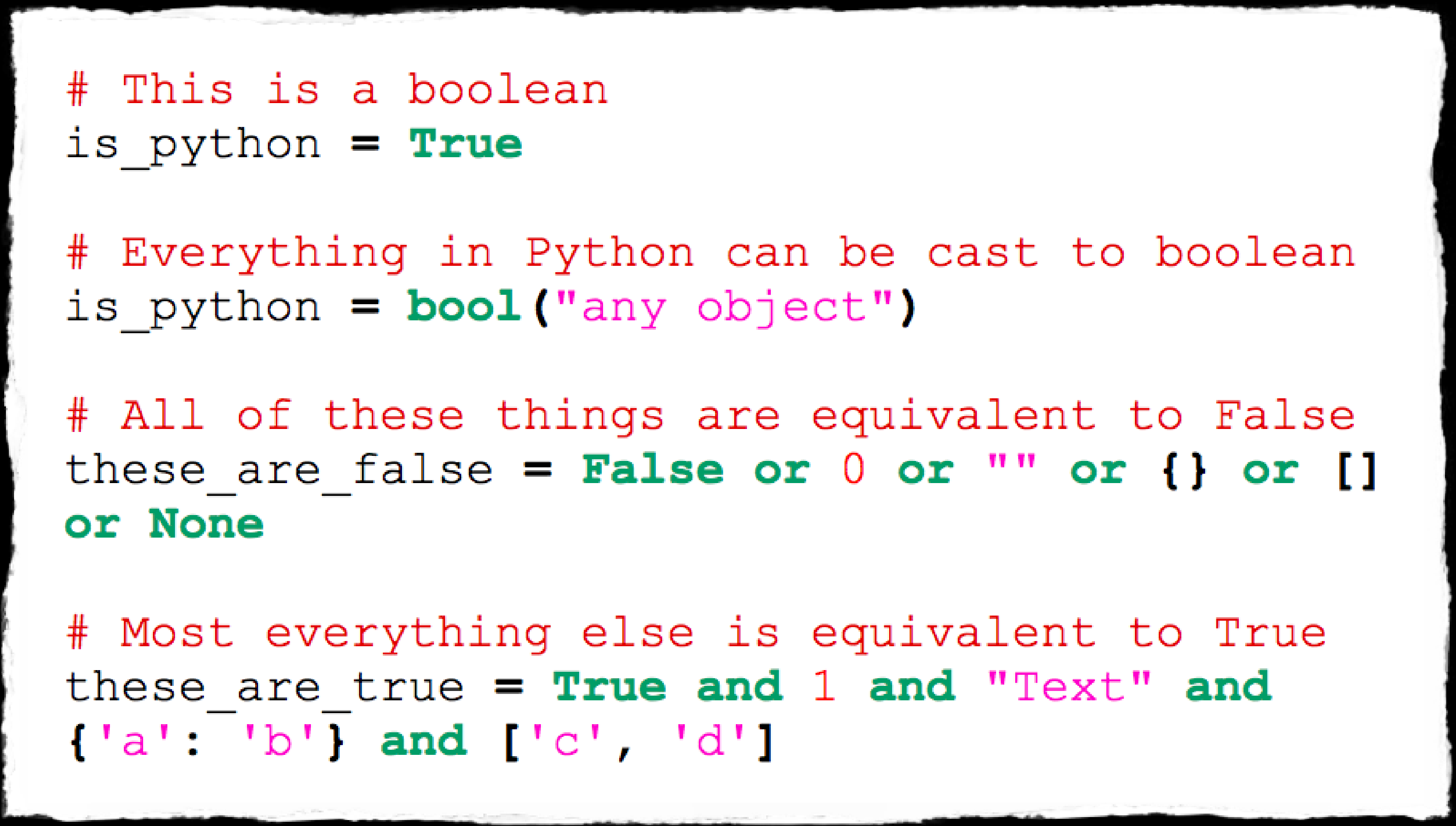
```python
person = {}

# Set by key / Get by key
person['name'] = 'Nowell Strite'

# Update
person.update({
    'favorites': [42, 'food'],
    'gender': 'male',
    })

# Any immutable object can be a dictionary key
person[42] = 'favorite number'
person[(44.47, -73.21)] = 'coordinates'
```

Dictionary methods



```
person = {'name': 'Nowell', 'gender': 'Male'}

person['name']
person.get('name', 'Anonymous')
# 'Nowell Strite'

person.keys()
# ['name', 'gender']

person.values()
# ['Nowell', 'Male']

person.items()
# [['name', 'Nowell'], ['gender', 'Male']]
```

Booleans



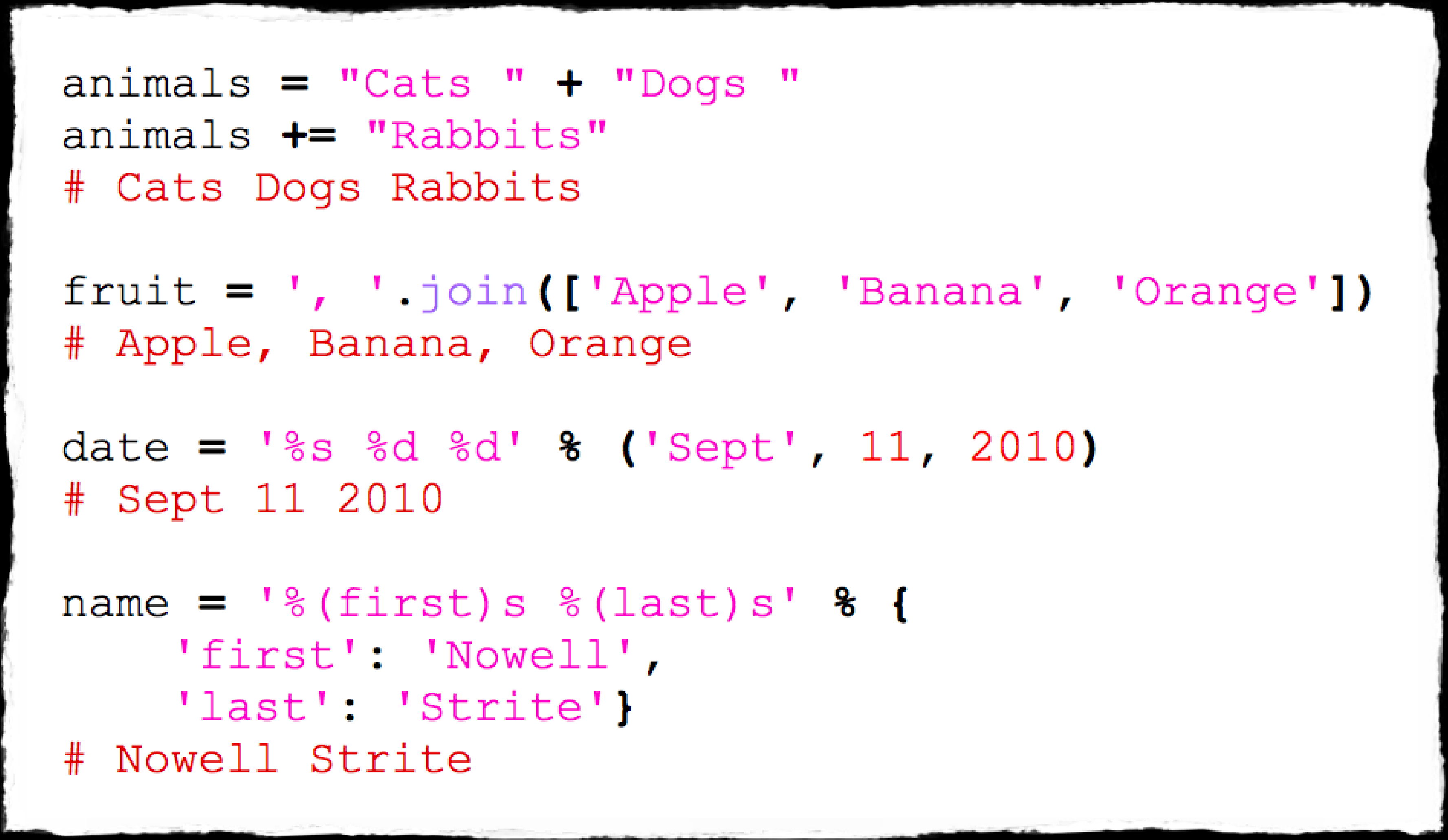
```python
# This is a boolean
is_python = True

# Everything in Python can be cast to boolean
is_python = bool("any object")

# All of these things are equivalent to False
these_are_false = False or 0 or "" or {} or []
or None

# Most everything else is equivalent to True
these_are_true = True and 1 and "Text" and
{'a': 'b'} and ['c', 'd']
```

Operators

Arithmetic

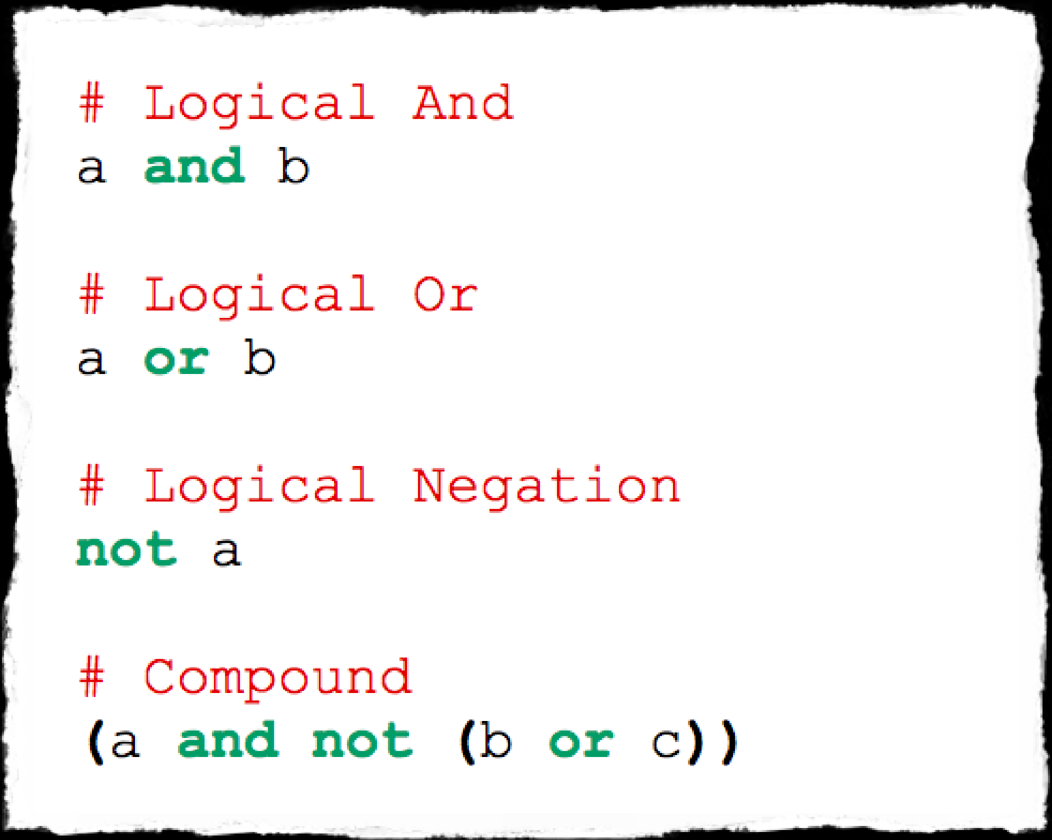
```
a = 10          # 10
a += 1          # 11
a -= 1          # 10

b = a + 1       # 11
c = a - 1       # 9

d = a * 2       # 20
e = a / 2       # 5
f = a % 3       # 1
g = a ** 2      # 100
```

String manipulation



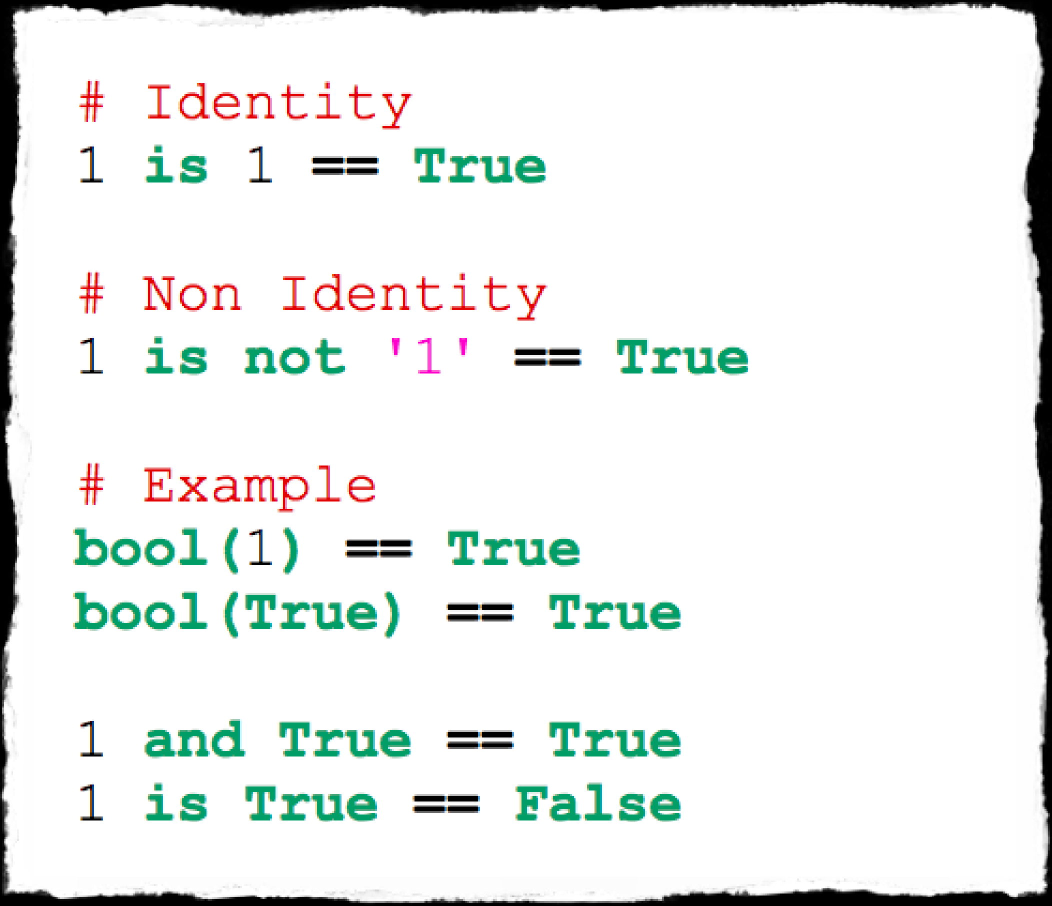
```python
animals = "Cats " + "Dogs "
animals += "Rabbits"
# Cats Dogs Rabbits

fruit = ', '.join(['Apple', 'Banana', 'Orange'])
# Apple, Banana, Orange

date = '%s %d %d' % ('Sept', 11, 2010)
# Sept 11 2010

name = '%(first)s %(last)s' % {
    'first': 'Nowell',
    'last': 'Strite'}
# Nowell Strite
```

Logical comparison



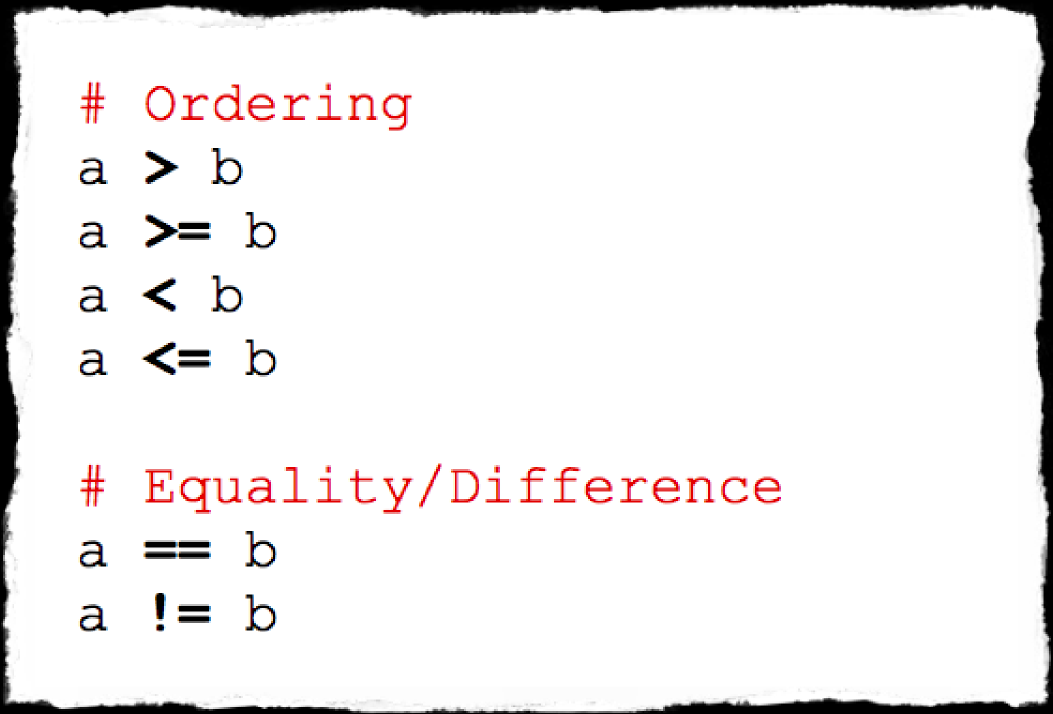
```
# Logical And
a and b

# Logical Or
a or b

# Logical Negation
not a

# Compound
(a and not (b or c))
```

Identity comparison



```
# Identity
1 is 1 == True

# Non Identity
1 is not '1' == True

# Example
bool(1) == True
bool(True) == True

1 and True == True
1 is True == False
```

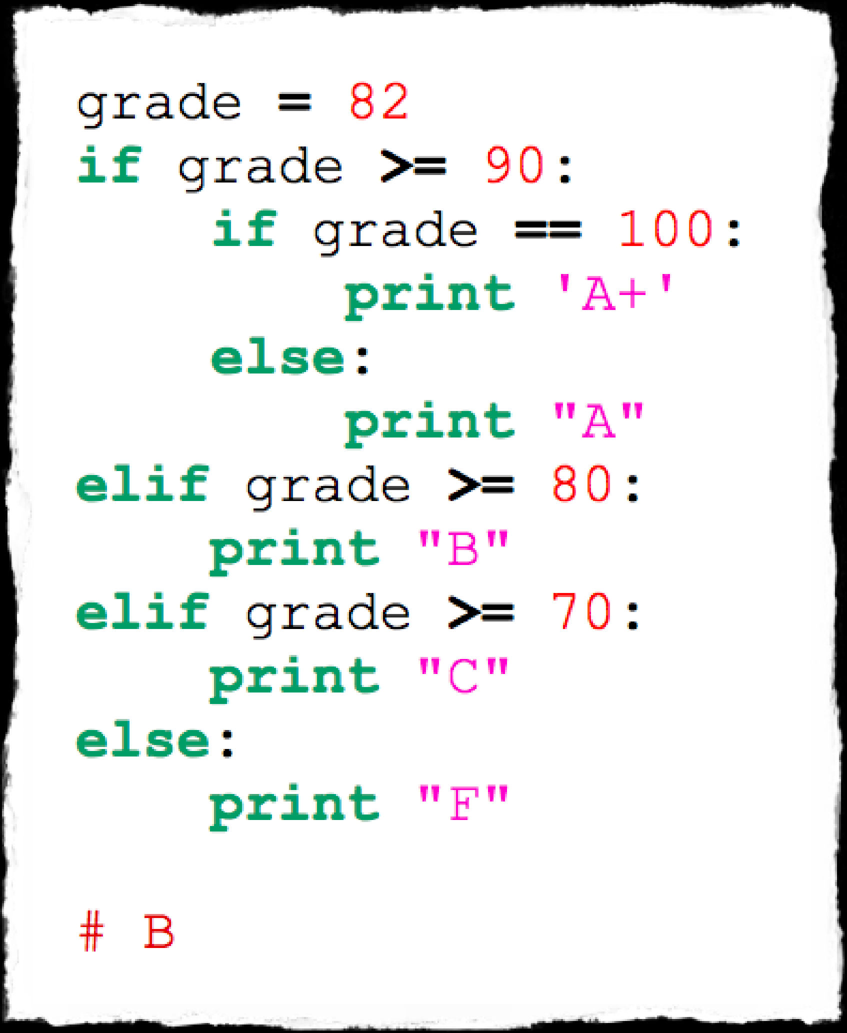
Arithmetic comparison



```
# Ordering
a > b
a >= b
a < b
a <= b

# Equality/Difference
a == b
a != b
```

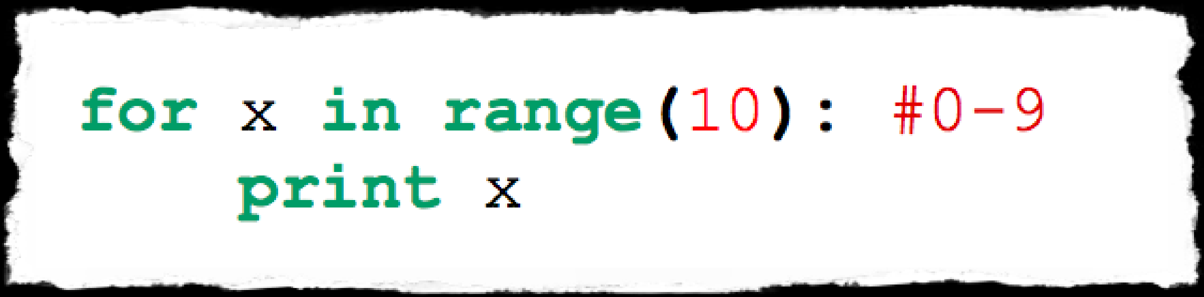
Control Flow

Conditionals

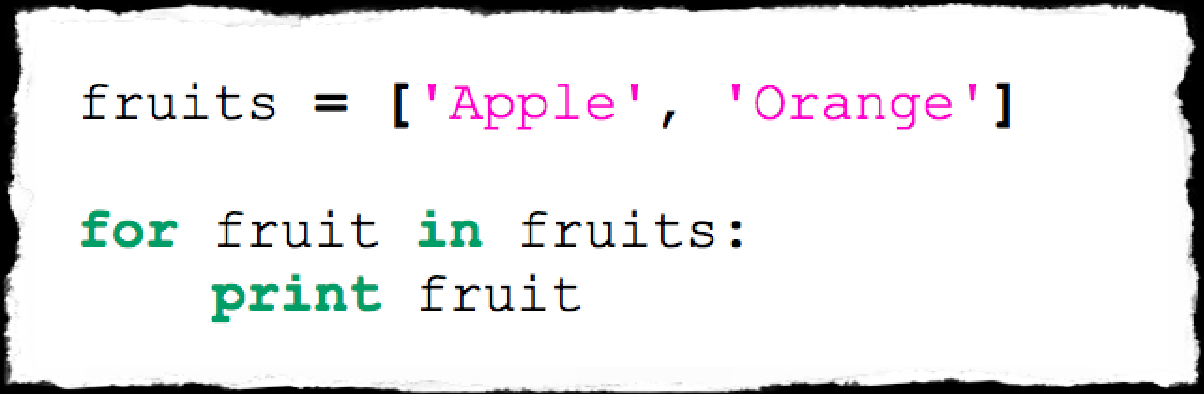


```
grade = 82
if grade >= 90:
    if grade == 100:
        print 'A+'
    else:
        print "A"
elif grade >= 80:
    print "B"
elif grade >= 70:
    print "C"
else:
    print "F"

# B
```

For loop

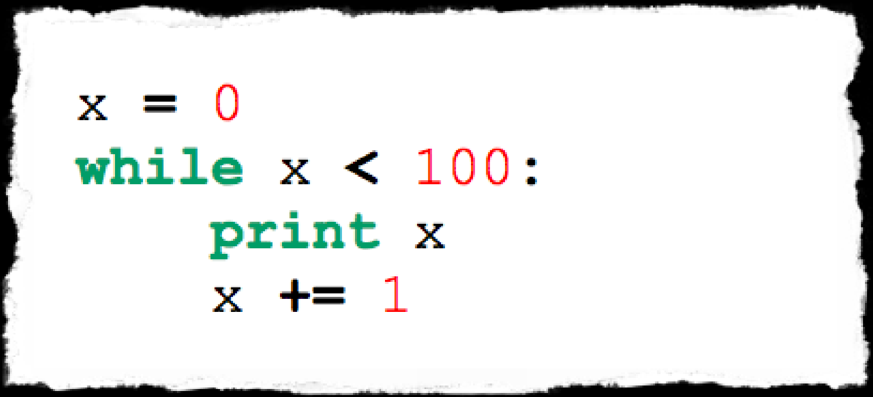


```python
for x in range(10): #0-9
    print x
```



```python
fruits = ['Apple', 'Orange']

for fruit in fruits:
    print fruit
```

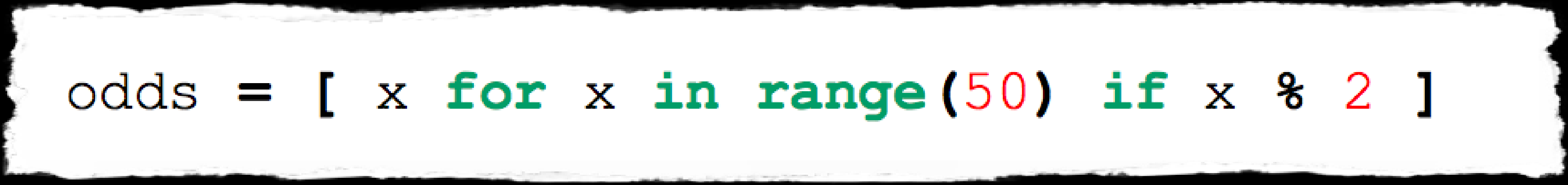
While loop



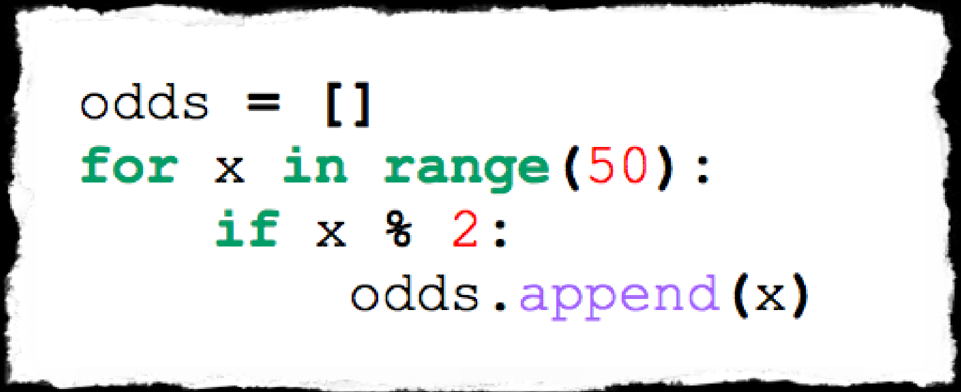
```
x = 0
while x < 100:
    print x
    x += 1
```

List comprehensions

Useful for replacing simple for-loops.



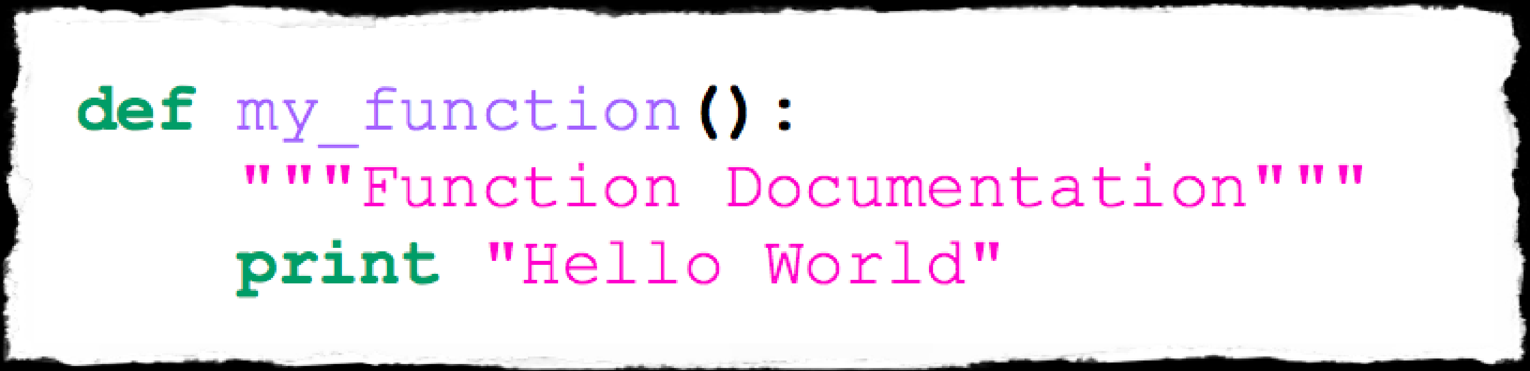
```python
odds = [ x for x in range(50) if x % 2 ]
```



```python
odds = []
for x in range(50):
    if x % 2:
        odds.append(x)
```

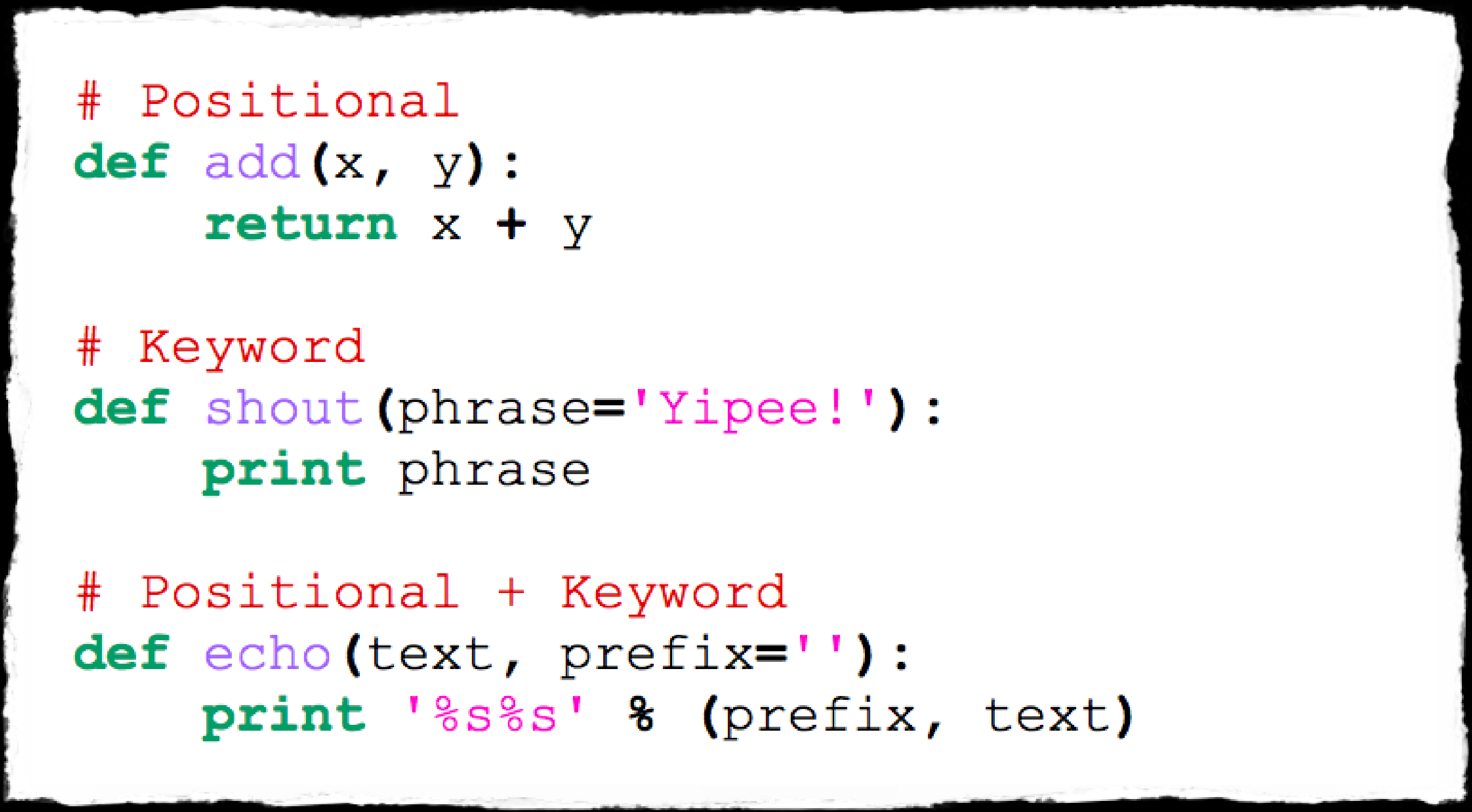
Functions

Basic function



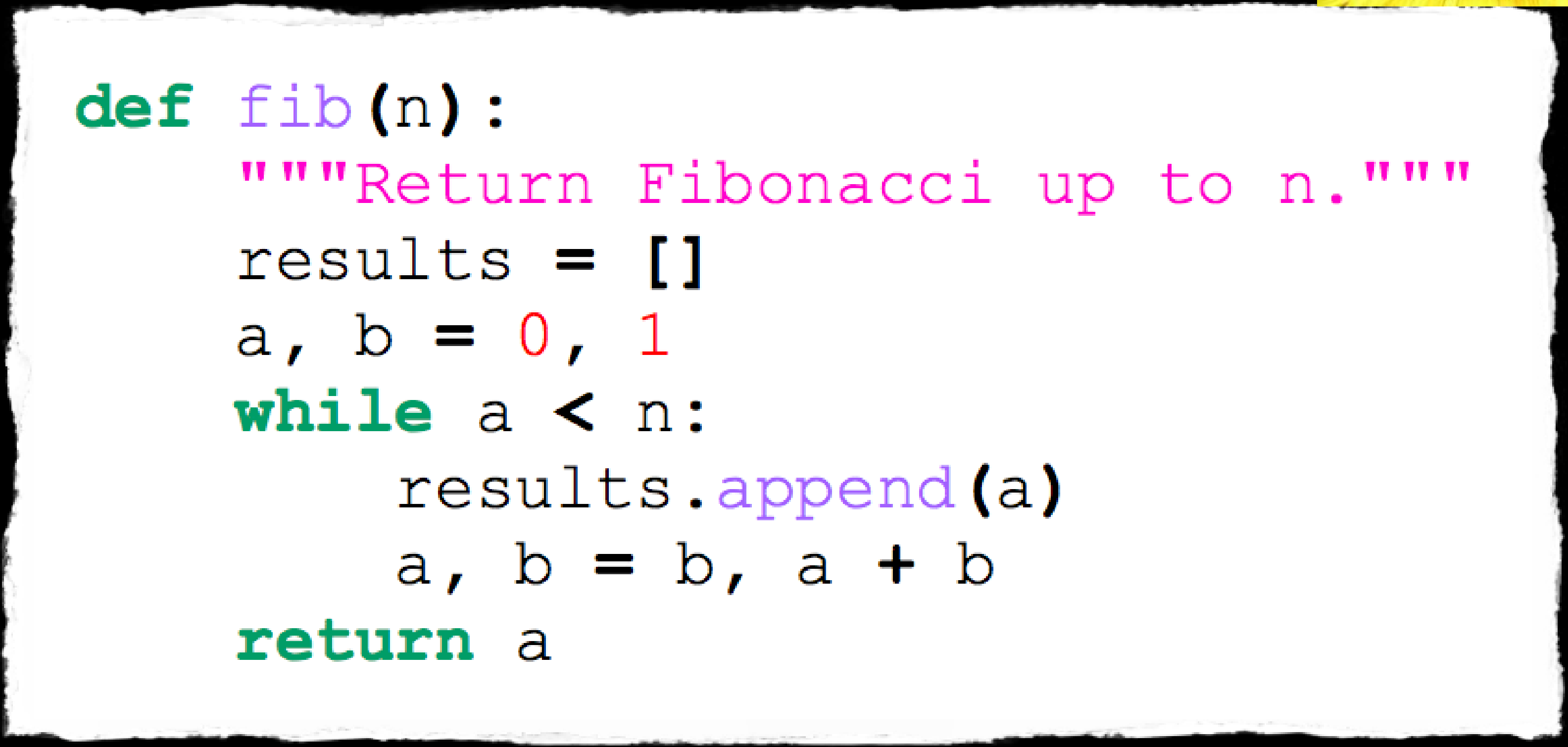
```
def my_function():
    """Function Documentation"""
    print "Hello World"
```

Function arguments



```
# Positional
def add(x, y):
    return x + y

# Keyword
def shout(phrase='Yipee!'):
    print phrase

# Positional + Keyword
def echo(text, prefix=''):
    print '%s%s' % (prefix, text)
```

Fibonacci



```
def fib(n):
    """Return Fibonacci up to n."""
    results = []
    a, b = 0, 1
    while a < n:
        results.append(a)
        a, b = b, a + b
    return a
```

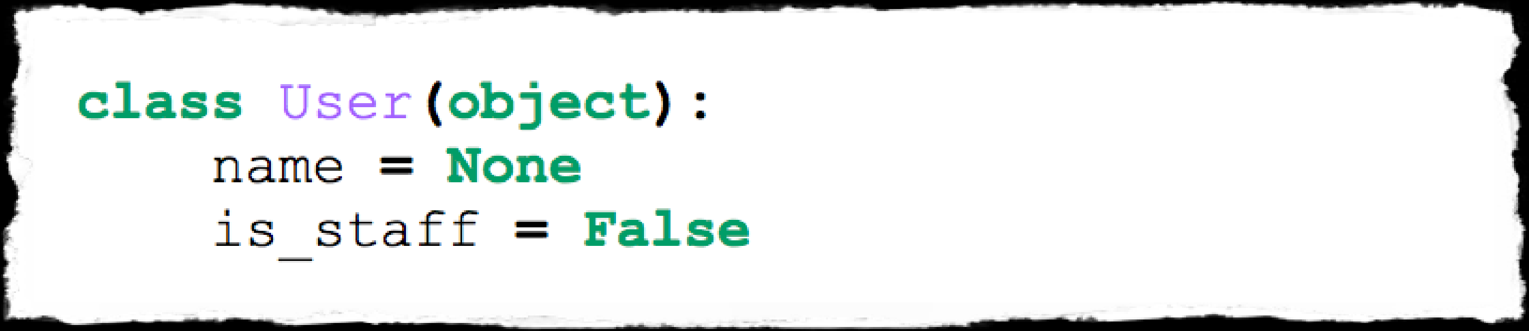
Classes

Class declaration

```
class User(object):
    pass
```

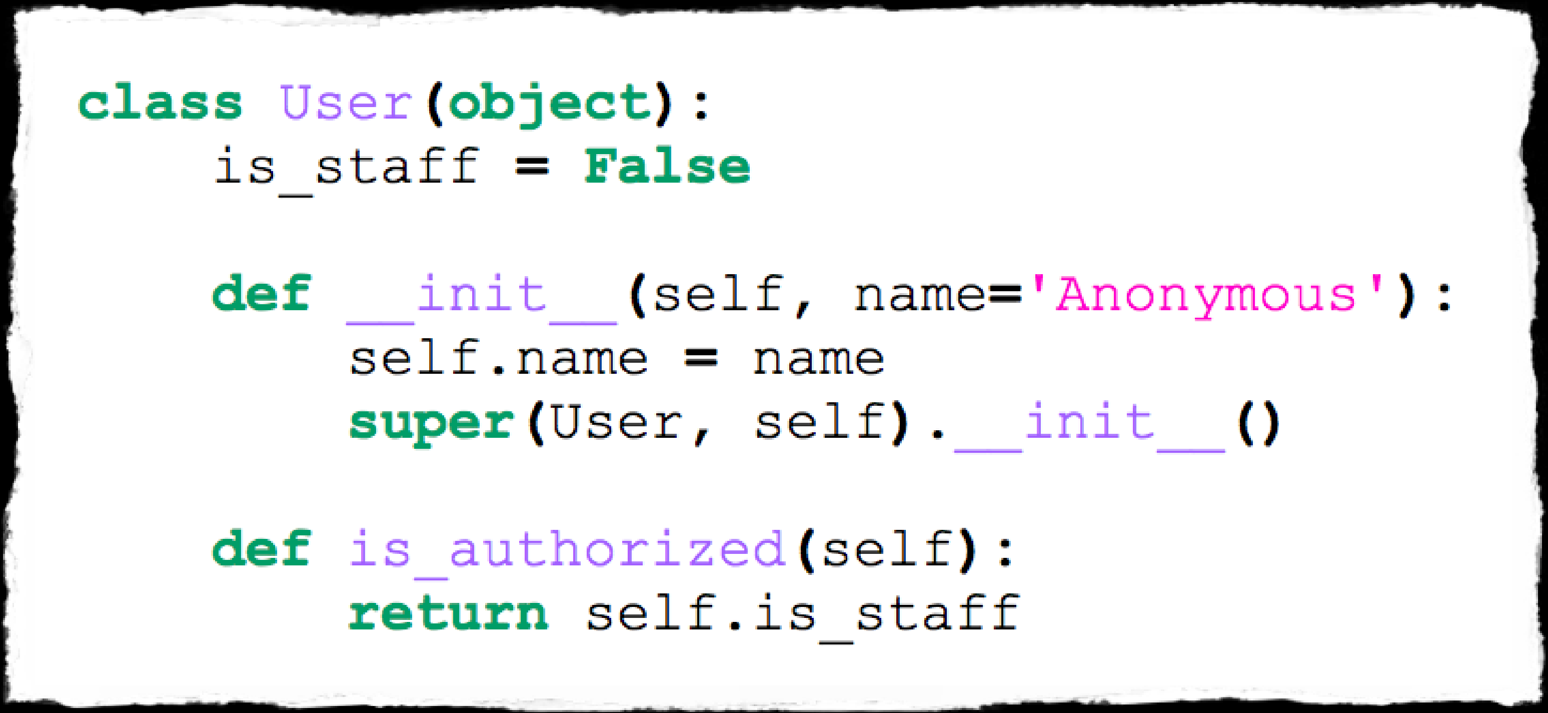
Class attributes

Attributes assigned at class declaration should always be immutable



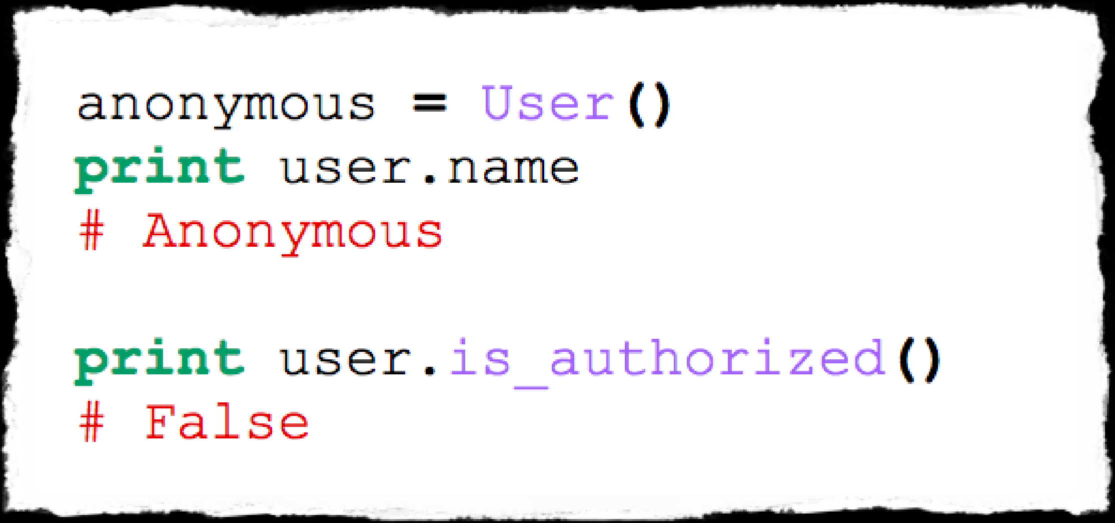
```
class User(object):
    name = None
    is_staff = False
```

Class methods



```
class User(object):
    is_staff = False

    def __init__(self, name='Anonymous'):
        self.name = name
        super(User, self).__init__()

    def is_authorized(self):
        return self.is_staff
```

Class instantiation & attribute access



```
anonymous = User()
print user.name
# Anonymous

print user.is_authorized()
# False
```

Modules

Usually big libraries that consist of a large number of classes and functions

Python's way

- No interfaces
- No real private attributes/functions
- Private attributes start (but do not end) with double underscores

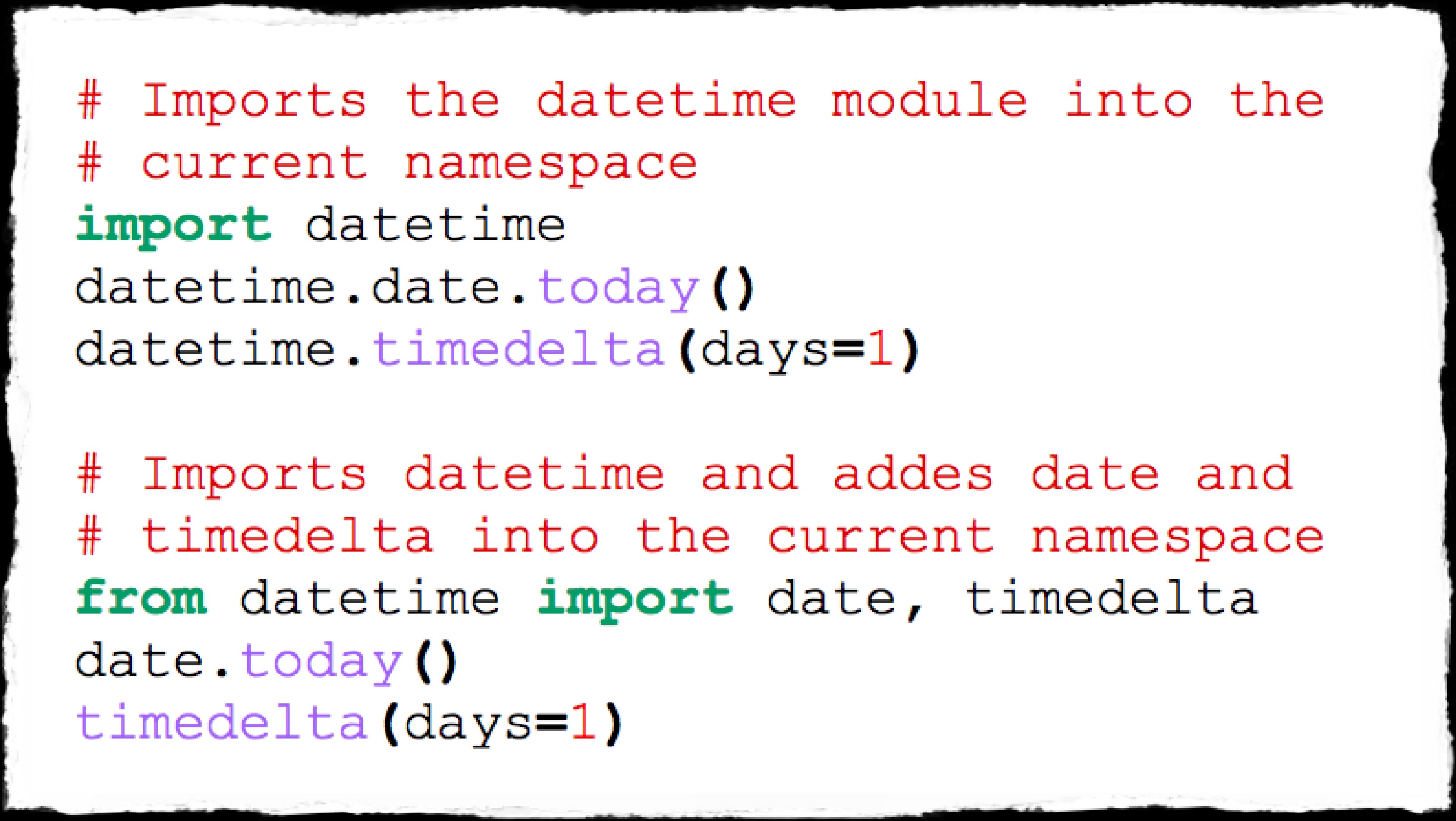
  __diff
- Special class methods start and end with double underscores

  __init__, __doc__, __cmp__, __str__

Imports

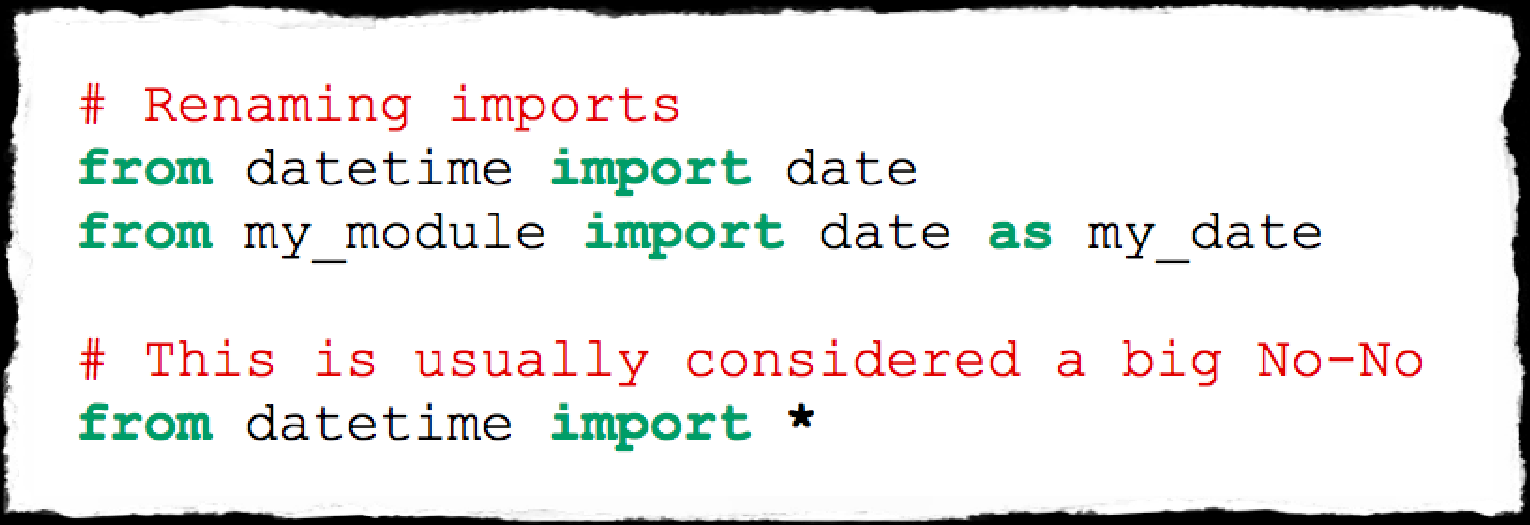
- Allows code isolation and re-use
- Adds references to variables/classes/functions/etc. into current namespace

Imports



```python
# Imports the datetime module into the
# current namespace
import datetime
datetime.date.today()
datetime.timedelta(days=1)

# Imports datetime and addes date and
# timedelta into the current namespace
from datetime import date, timedelta
date.today()
timedelta(days=1)
```

More imports



```python
# Renaming imports
from datetime import date
from my_module import date as my_date

# This is usually considered a big No-No
from datetime import *
```

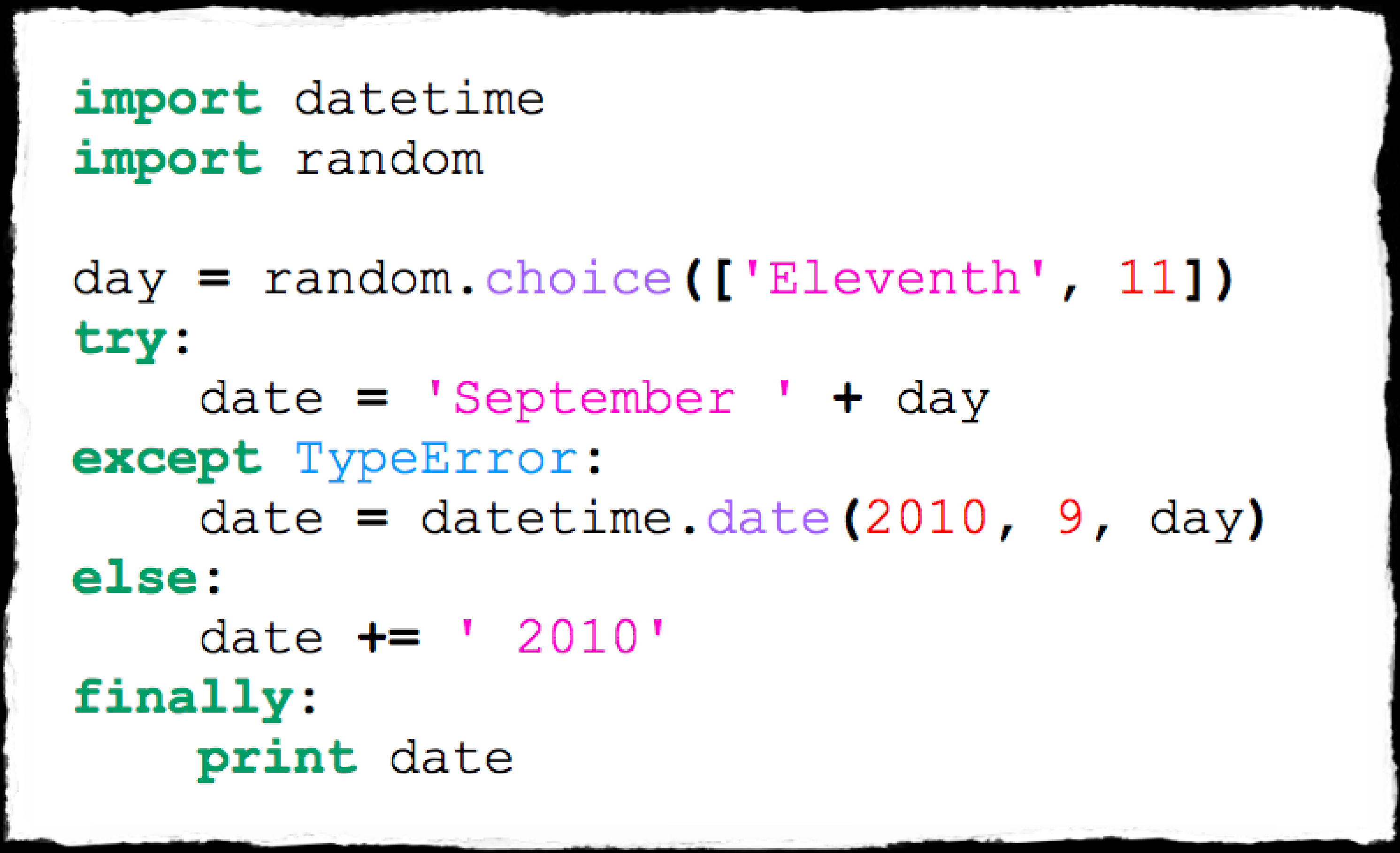
Useful modules/built-in libraries

- *os*, *sys and shutil* – general computer handling
- *datetime* – handling date and time
- *webbrowser* – url handling
- *ftplib* – remote ftp-server reading/writing
- *xarrays* – handling of netCDF files

Useful modules/libraries

- *numpy* and *scipy* – make python be like “matlab”. numpy arrays are very suitable for rasters
- *matplotlib* – a plotting library
- *pandas* – easy-to-use data structures and data analysis tools
- *PySAL* – suite of spatial analytical methods
- *PySolar* – for sun applications

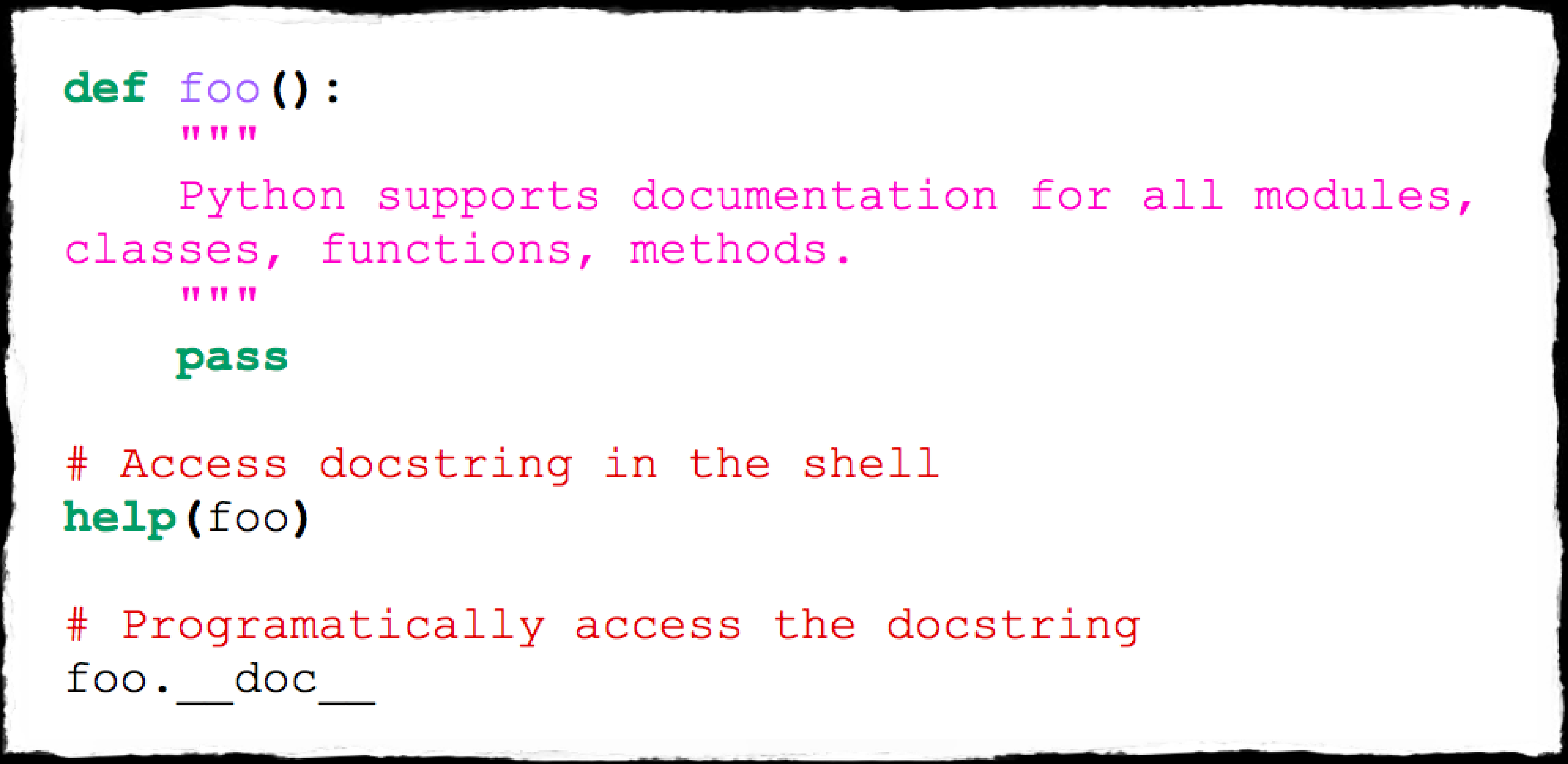
Error handling



```python
import datetime
import random

day = random.choice(['Eleventh', 11])
try:
    date = 'September ' + day
except TypeError:
    date = datetime.date(2010, 9, day)
else:
    date += ' 2010'
finally:
    print date
```

Documentation

Docstrings



```
def foo():
    """
    Python supports documentation for all modules,
classes, functions, methods.
    """
    pass

# Access docstring in the shell
help(foo)

# Programatically access the docstring
foo.__doc__
```

Tools

Integrated Development Environment (IDE)

- Spyder (Conda)
- The Jupyther Notebook
- Komodo
- PyCharm
- Eclipse (PyDev)
- Visual Studio Code (VSCode)

Resources

- http://python.org/
- http://diveintopython.org/

- For Geoscience: https://geo-python-site.readthedocs.io/