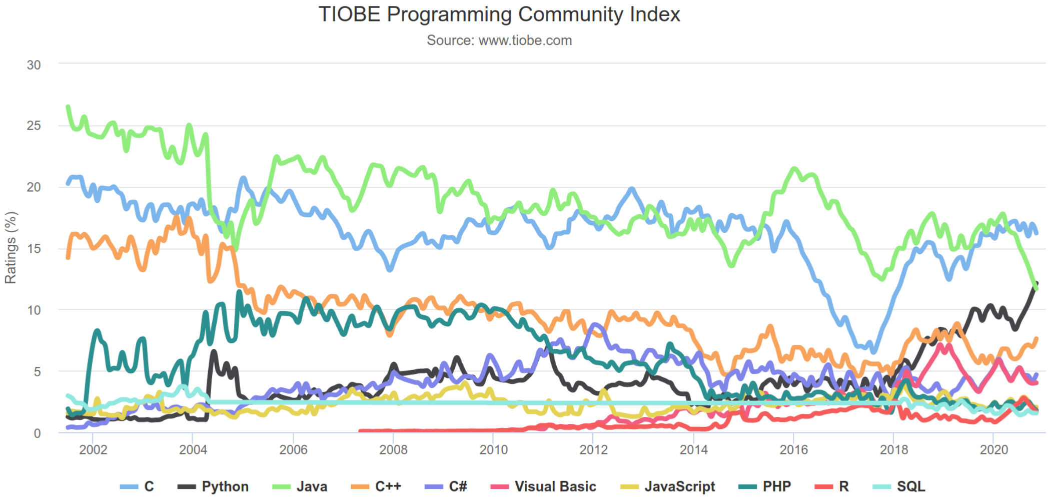
Python Introduction

Enigma Webinar

Aslak Johansen asjo@mmmi.sdu.dk

November 18, 2020

Introduction



Philosophy

- Scripting language.
- Emphasis on code readability.
- Execution speed is not a priority.
- Language simplicity is a priority.
- Readability is a priority.
- Blocks are defined through indentation.

Philosophy

- Scripting language.
- Emphasis on code readability.
- Execution speed is not a priority.
- Language simplicity is a priority.
- Readability is a priority.
- Blocks are defined through indentation.

Zen of Python `https://en.wikipedia.org/wiki/Zen_of_Python`.

Philosophy

- Scripting language.
- Emphasis on code readability.
- Execution speed is not a priority.
- Language simplicity is a priority.
- Readability is a priority.
- Blocks are defined through indentation.

Zen of Python `https://en.wikipedia.org/wiki/Zen_of_Python`.

Quoting Monty Python is officially considered good style.

Characteristics

- ***Interpreted*** interpreter + code → execution
  Java: compiler + code → bytecode; bytecode + interpreter → execution
  C: compiler + code → code → execution
- ***Garbage Collected*** Makes use of automatic memory management (no need to keep tract of when to *free* individual allocations).
- ***Dynamically Typed*** Type safety is verified at runtime.
- ***Late Binding*** Names are looked up at runtime.
- ***High-Level*** Focus is on serving as a good language for abstract (as in: machine distant) operations.

Good Matches

- String manipulation
- Data transformation
- Batch data processing
- Filesystem traversal
- Quick and dirty coding
- Calculator

Bad Matches

Bad Matches

- The use of a global interpreter lock (GIL) makes the language a bad fit for any problem that would benefit from concurrency.

Bad Matches

- The use of a global interpreter lock (GIL) makes the language a bad fit for any problem that would benefit from concurrency.
- The lack of "compile-time" type checking/safely makes the language a bad fit for any problem that requires robustness or is long-running.

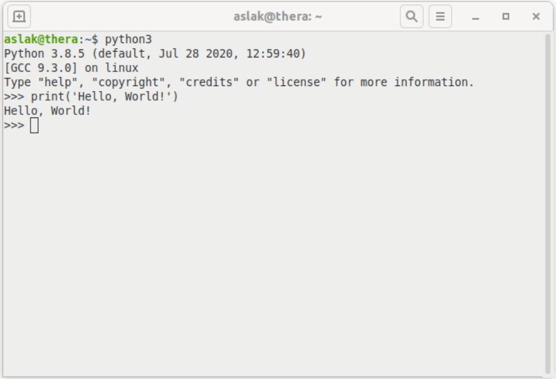
Bad Matches

- The use of a global interpreter lock (GIL) makes the language a bad fit for any problem that would benefit from concurrency.
- The lack of "compile-time" type checking/safely makes the language a bad fit for any problem that requires robustness or is long-running.
- The level of abstraction makes python a bad fit for any computationally heavy operations.

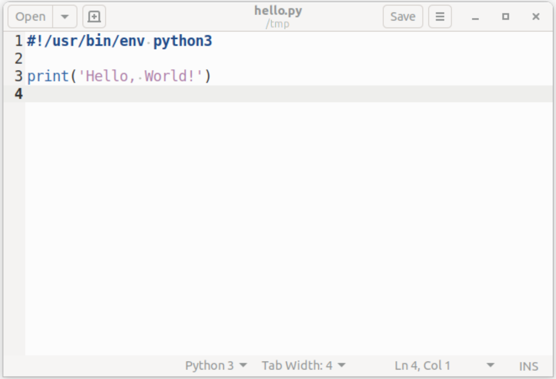
Bad Matches

- The use of a global interpreter lock (GIL) makes the language a bad fit for any problem that would benefit from concurrency.
- The lack of "compile-time" type checking/safely makes the language a bad fit for any problem that requires robustness or is long-running.
- The level of abstraction makes python a bad fit for any computationally heavy operations.
  - **Note:** Python code can call C or Fortran code which can do the heavy lifting. This is what `numpy` does.

Two Modes of Execution



```
aslak@thera:~$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello, World!')
Hello, World!
>>> 
```



```
#!/usr/bin/env python3

print('Hello, World!')

```

Python 3 Tab Width: 4 Ln 4, Col 1 INS

Part 1:
Getting Started

Imports

```
import os
from sys import argv
from sys import exit as bye

print(os.name)
bye()
```

First Steps

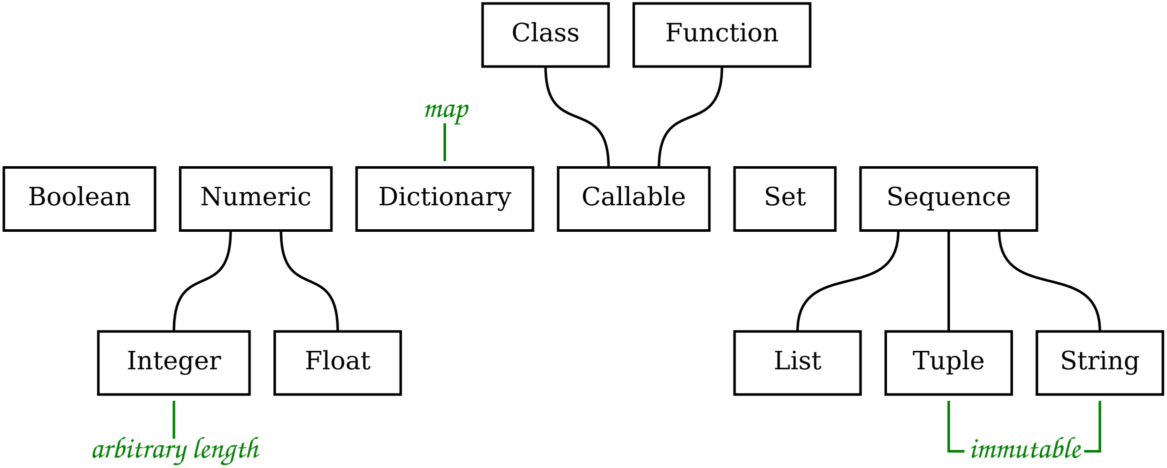
```
#!/usr/bin/env python3
import sys

print("Hello, world!")
sys.exit() # this is really not necessary
```

Command-Line Arguments

```python
from sys import argv, exit

if len(argv) != 3:
    print('Syntax: %s INPUT_FILENAME OUTPUT_FILENAME' % argv[0])
    print('        %s log.txt analysis.csv' % argv[0])
    exit(1)
input_filename  = argv[1]
output_filename = argv[2]

print('%s -> %s' % (input_filename, output_filename))
```

Part 2:
Basic Datatypes and -Structures

Types


Class
Function
map
Boolean
Numeric
Dictionary
Callable
Set
Sequence
Integer
Float
List
Tuple
String
arbitrary length
immutable

Boolean Operators

In python boolean operators are spelled out.

```
if page < pagecount and good_book:
    print('Enjoy!')

while not there:
    print('Are we there yet?')

if finished or not started:
    print('Not much is happening :-(')
```

String Operations

```
>>> s1 = "Alice was beginning to get very tired of sitting by her sister on the bank"
>>> s1
'Alice was beginning to get very tired of sitting by her sister on the bank'
>>> s2 = 'and of having nothing to do'
>>> s2
'and of having nothing to do'
>>> s = s1+", "+s2
>>> s
'Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do'
>>> s = s.replace(',', '')
>>> s
'Alice was beginning to get very tired of sitting by her sister on the bank and of having nothing to do'
>>> words = s.split(' ')
>>> words
['Alice', 'was', 'beginning', 'to', 'get', 'very', 'tired', 'of', 'sitting', 'by', 'her', 'sister', 'on', 'the',
'bank', 'and', 'of', 'having', 'nothing', 'to', 'do']
>>> '%d: %s' % (3, 'March')
'3: March'
>>> '_'.join(words)
'Alice_was_beginning_to_get_very_tired_of_sitting_by_her_sister_on_the_bank_and_of_having_nothing_to_do'
```

Functions

```
def add (a, b, c=0):
    return a+b+c

print(add(1,2,3))
print(add(1,2))

a = add
print(a(1,2))
```

Functions

```
def add (a, b, c=0):
    return a+b+c

print(add(1,2,3))
print(add(1,2))

a = add
print(a(1,2))
```

```
6
3
3
```

Type Introspection

```
>>> t = type(True)
>>> t
<class 'bool'>
>>> type(t)
<class 'type'>
>>> type(bool)
<class 'type'>
>>> t == bool
True
```

Type Introspection

```
>>> def fun(var): return var
...
>>> type(fun)
<class 'function'>
>>> f = fun
>>> type(f)
<class 'function'>
>>> f(1)
1
>>> g = lambda a: a
>>> type(g)
<class 'function'>
>>> g(1)
1
```

Object-Orientation

```
class Person:
    def __init__ (self, name):
        self.name = name

    def get_name (self): return self.name
    def set_name (self, name):
        self.name = name

def printname (self):
    print(self.name)

p = Person('Aslak')
print(p.get_name())

setattr(Person, 'printname', printname)
p.printname()
```

Part 3:
Flow Control

Branching

```
if len(lines)>0 and len(line[0])>0 and line[0][0]=='#':
    print('First line is a coment')

parts = line.split(' ')
command = parts[0]
if   command=='load':
    load_file()
elif command=='save':
    save_file()
elif command=='quit':
    quit()
else:
    print('Unknown command "'+command+'"')
```

Missing For-Loop

Python does not have a *for* loop.

Missing For-Loop

Python does not have a *for* loop.

Python has a *foreach* loop.

Missing For-Loop

Python does not have a *for* loop.

Python has a *foreach* loop.

Iterating over a list:

```
for line in lines:
    print(line)
```

Missing For-Loop

Python does not have a *for* loop.

Python has a *foreach* loop.

Iterating over a list:

```
for line in lines:
    print(line)
```

Iterating over a list with access to the index:

```
for i in range(len(lines)):
    line = lines[i]
    print(str(i)+': '+line)
```

Generating Ranges of Integers

The `range` function returns a generator for a sequence of integers.

Generating Ranges of Integers

The `range` function returns a generator for a sequence of integers.

```
>>> range(5)
range(0, 5)
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(1,5))
[1, 2, 3, 4]
>>> list(range(1,5,2))
[1, 3]
>>> for i in range(1,5,2):
...     print(i)
...
1
3
```

Part 4:
Lists

List Operations

```
>>> l = [1,2,3]
>>> l
[1, 2, 3]
>>> l.append(4)
>>> l
[1, 2, 3, 4]
>>> l.extend([7,6,5])
>>> l
[1, 2, 3, 4, 7, 6, 5]
>>> sorted(l)
[1, 2, 3, 4, 5, 6, 7]
>>> l
[1, 2, 3, 4, 7, 6, 5]
>>> l.sort()
>>> l
[1, 2, 3, 4, 5, 6, 7]
>>> len(l)
7
>>> l[2], l[-1]
(3, 7)
>>> l[2:]
[3, 4, 5, 6, 7]
>>> l[2:4]
[3, 4]
>>> l[:4]
[1, 2, 3, 4]
>>> 4 in l, 42 in l
(True, False)
```

Part 5:
Dictionaries

Dictionary Operations

```
>>> {}
{}
>>> d = {'jan': 1, 'feb': 2, 'mar': 3}
>>> d
{'jan': 1, 'feb': 2, 'mar': 3}
>>> d['jan']
1
>>> d['apr'] = 4
>>> d
{'jan': 1, 'feb': 2, 'mar': 3, 'apr': 4}
>>> d['list'] = [1,2,3]
>>> d
{'jan': 1, 'feb': 2, 'mar': 3, 'apr': 4, 'list': [1, 2, 3]}
>>> 'jan' in d
True
>>> 'may' in d
False
>>> d.keys()
dict_keys(['jan', 'feb', 'mar', 'apr', 'list'])
>>> list(d.keys())
['jan', 'feb', 'mar', 'apr', 'list']
>>> for key in d: print(key)
...
jan
feb
mar
apr
list
>>> del(d['feb'])
>>> d
{'jan': 1, 'mar': 3, 'apr': 4, 'list': [1, 2, 3]}
```

Part 6:
Strings

Strings: Basic Operations

```
>>> initial = ' once upon a time  '
>>> initial
' once upon a time  '
>>> len(initial)
19
>>> stripped = initial.strip()
>>> stripped
'once upon a time'
>>> words = stripped.split(' ')
>>> words
['once', 'upon', 'a', 'time']
>>> words[1], stripped[1]
('upon', 'n')
>>> joined = '_'.join(words)
>>> joined
'once_upon_a_time'
```

Regular Expressions

```python
import re

urls = [
    'https://www.gutenberg.org/files/11/11-h/11-h.htm',
    'https://golang.org',
    'http://www.google.com:80/',
    'definitely not a URL',
]
pattern = re.compile('([^:]+)://([^:/]+)(:\d+|)(/.*|)')

for url in urls:
    mo = pattern.match(url)

    if mo:
        print('proto="%s" domain="%s" port="%s" path="%s"' %
              (mo.group(1), mo.group(2), mo.group(3), mo.group(4)))
```

Regular Expressions

```
import re

urls = [
    'https://www.gutenberg.org/files/11/11-h/11-h.htm',
    'https://golang.org',
    'http://www.google.com:80/',
    'definitely not a URL',
]
pattern = re.compile('([^:]+)://([^:/]+)(:\d+|)(/.*|)')

for url in urls:
    mo = pattern.match(url)

    if mo:
        print('proto="%s" domain="%s" port="%s" path="%s"' %
              (mo.group(1), mo.group(2), mo.group(3), mo.group(4)))
```

```
proto="https" domain="www.gutenberg.org" port="" path="/files/11/11-h/11-h.htm"
proto="https" domain="golang.org" port="" path=""
proto="http" domain="www.google.com" port=":80" path="/"
```

Part 7:
Files

Files: Basics

```
fo = open(filename, mode)
# do something with 'fo'
fo.close()
```

Files: Basics

```
fo = open(filename, mode)
# do something with 'fo'
fo.close()
```

Modes:

- `’r’` read mode **(default ⇒ may be omitted)**
- `’w’` write mode (clears file on open)
- `’a’` append mode (continues at end of file)

Files: Basics

```
fo = open(filename, mode)
# do something with 'fo'
fo.close()
```

Modes:

- ’r’ read mode **(default ⇒ may be omitted)**
- ’w’ write mode (clears file on open)
- ’a’ append mode (continues at end of file)

Read using:

```
lines = fo.readlines()
```

Files: Basics

```
fo = open(filename, mode)
# do something with 'fo'
fo.close()
```

Modes:

- ’r’ read mode **(default ⇒ may be omitted)**
- ’w’ write mode (clears file on open)
- ’a’ append mode (continues at end of file)

Read using:

```
lines = fo.readlines()
```

Write using:

```
fo.writelines(lines)
```

Files: Implicit Closing of File Objects

```
with open(input_filename) as fo:
    lines = fo.readlines()

with open(output_filename, 'w') as fo:
    fo.writelines(lines)
```

Files: Reading from a Stream of Lines

```
with open(filename) as fo:
    for line in fo:
        print(line)
```

Files: Directories

```
>>> from os import listdir, path
>>> listdir('/')
['var', 'etc', 'run', 'proc', 'media', 'home', 'sbin', 'srv', 'lost+found',
'boot', 'tmp', 'sys', 'lib32', 'bin', 'dev', 'snap', 'cdrom', 'libx32',
'root', 'usr', 'lib', 'lib64', 'opt', 'mnt']
>>> listdir(path.sep.join(['', 'usr']))
['src', 'local', 'sbin', 'libexec', 'share', 'lib32', 'bin', 'libx32', 'lib',
'lib64', 'include', 'games']
>>> path.isfile('/etc')
False
>>> path.isdir('/etc')
True
>>> path.exists('/etc')
True
>>> path.islink('/etc')
False
>>> import os
>>> os.mkdir('/tmp/dir')
>>> os.unlink('/tmp/somefile')
```

Files: Process Interaction

```python
from subprocess import Popen, STDOUT, PIPE

def system (command, err=STDOUT, out=PIPE):
    p = Popen(command, shell=True, stderr=err, stdout=out)
    output = p.communicate()[0]
    return output.decode('utf-8')

output = system('ls /')
print(output)
```

Files: Process Interaction

```python
from subprocess import Popen, STDOUT, PIPE

def system (command, err=STDOUT, out=PIPE):
    p = Popen(command, shell=True, stderr=err, stdout=out)
    output = p.communicate()[0]
    return output.decode('utf-8')

output = system('ls /')
print(output)
```

```
$ python3 processexample.py
bin
boot
cdrom
dev
etc
home
lib
...
```

Part 8:
Marshalling and JSON

Marshalling and JSON

```
>>> import json
>>> data = {'a': 1, 'b': [1,2,3], 'c': 'abc'}
>>> data
{'a': 1, 'b': [1, 2, 3], 'c': 'abc'}

>>> json.dumps(data)
'{"a": 1, "b": [1, 2, 3], "c": "abc"}'
>>> s = json.dumps(data, sort_keys=True, indent=4, separators=(',', ': '))
>>> s
'{\n    "a": 1,\n    "b": [\n        1,\n        2,\n        3\n    ],\n    "c": "abc"\n}'
>>> print(s)
{
    "a": 1,
    "b": [
        1,
        2,
        3
    ],
    "c": "abc"
}

>>> data2 = json.loads(s)
>>> data2
{'a': 1, 'b': [1, 2, 3], 'c': 'abc'}
```

Part 9:
Higher-Order Functions

Higher-Order Functions

```
>>> l = [-17,2,5,-4,4,7,-3,-1,9,1]
>>> incr = lambda v: v+1
>>> map(incr, l)
<map object at 0x7f33c8440b20>
>>> list(map(incr, l))
[-16, 3, 6, -3, 5, 8, -2, 0, 10, 2]
>>> pos = lambda v: v>=0
>>> filter(pos, l)
<filter object at 0x7f33c8440b20>
>>> list(filter(pos, l))
[2, 5, 4, 7, 9, 1]
>>> list(map(incr, filter(pos, l)))
[3, 6, 5, 8, 10, 2]
```

Part 10:
Working in Modules

Working in Modules

`awesome.py` (the module)

```python
def add (a, b):
    return a+b
```

Working in Modules

awesome.py (the module)

```
def add (a, b):
    return a+b
```

main.py (the consumer of the module)

```
import awesome
print(awesome.add(1,2))
```

Working in Modules

awesome.py (the module)

```python
def add (a, b):
    return a+b
```

main.py (the consumer of the module)

```python
import awesome
print(awesome.add(1,2))
```

Result of execution

```
$ python3 main.py
3
```

Part 11:
Pitfalls

Mutable Default Arguments

```python
def incr (increment=1, data=[], newvalues=[]):
    for value in newvalues:
        data.append(value)
    for i in range(len(data)):
        data[i] += increment
    return data

print(incr(newvalues=[1]))
print(incr(newvalues=[1]))
```

Mutable Default Arguments

```python
def incr (increment=1, data=[], newvalues=[]):
    for value in newvalues:
        data.append(value)
    for i in range(len(data)):
        data[i] += increment
    return data

print(incr(newvalues=[1]))
print(incr(newvalues=[1]))
```

```
$ python3 mutabledefaults.py
[2]
[3, 2]
```

Scope and Globals

```
a = 42
b = 56

def fun (value):
    global b

    a = value
    b = value
    print(a, b)

fun(-1)
print(a, b)
```

Scope and Globals

```python
a = 42
b = 56

def fun (value):
    global b

    a = value
    b = value
    print(a, b)

fun(-1)
print(a, b)
```

```
$ python3 globals.py
-1 -1
42 -1
```

Part 12:
Next Steps

SymPy

```
import sympy

s = input('Please enter an expression over i and j: ')
expr = sympy.sympify(s)
symi = sympy.symbols('i')
symj = sympy.symbols('j')

print('expr='+str(expr))
print('diff(expr, i)='+str(sympy.diff(expr, symi)))

for i in range(10):
    j = i % 3
    v = expr.subs([(symi, i), (symj, j)])
    print('i=%d j=%d => result=%f' % (i, j, v))
```

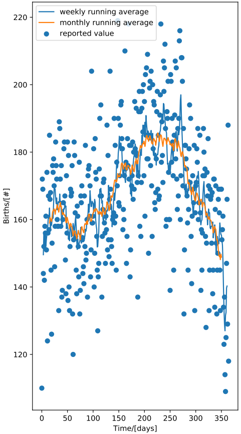
SymPy

```python
import sympy

s = input('Please enter an expression over i and j: ')
expr = sympy.sympify(s)
symi = sympy.symbols('i')
symj = sympy.symbols('j')

print('expr='+str(expr))
print('diff(expr, i)='+str(sympy.diff(expr, symi)))

for i in range(10):
    j = i % 3
    v = expr.subs([(symi, i), (symj, j)])
    print('i=%d j=%d => result=%f' % (i, j, v))
```

```
Please enter an expression over i and j: 3*i*i+4.83*i+j
expr=3*i**2 + 4.83*i + j
diff(expr, i)=6*i + 4.83
i=0 j=0 => result=0.000000
i=1 j=1 => result=8.830000
i=2 j=2 => result=23.660000
i=3 j=0 => result=41.490000
i=4 j=1 => result=68.320000
i=5 j=2 => result=101.150000
i=6 j=0 => result=136.980000
i=7 j=1 => result=181.810000
i=8 j=2 => result=232.640000
i=9 j=0 => result=286.470000
```

Pandas

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('dkbirths2019.csv', names=['day', 'raw'])
df['avgw'] = df['raw'].rolling(window=7, center=True).mean()
df['avgm'] = df['raw'].rolling(window=30, center=True).mean()

plt.figure(figsize=[5,10])
plt.xlabel('Time/[days]')
plt.ylabel('Births/[#]')
plt.scatter(df['day'], df['raw'],label='reported value')
plt.plot(df['day'], df['avgw'],label='weekly running average')
plt.plot(df['day'], df['avgm'],label='monthly running average')
plt.legend(loc=2)
plt.savefig('running-average.pdf')
```



AsyncIO

```
import asyncio
from aiohttp import web

async def handler (request):
    method  =         request.method
    path    = '/'+str(request.rel_url)[1:]
    payload =   await request.content.read()

    try:
        with open(path) as fo:
            return web.Response(status=200, text=''.join(fo.readlines()))
    except:
        return web.Response(status=500, text=str([method, path]))

async def main(interface, port):
    proto  = web.Server(handler)
    server = await loop.create_server(proto, interface, port)

loop = asyncio.get_event_loop()
asyncio.Task(main('0.0.0.0', 8080))

# enter service loop
try:
    loop.run_forever()
except KeyboardInterrupt:
    print('')
    print('STATUS: Exiting ...')
    loop.close()
    exit(0)
```

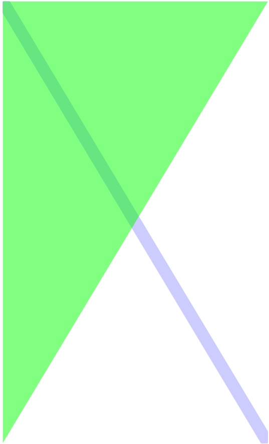
AsyncIO

```
import asyncio
from aiohttp import web

async def handler (request):
    method  =         request.method
    path    = '/'+str(request.rel_url)[1:]
    payload =   await request.content.read()

    try:
        with open(path) as fo:
            return web.Response(status=200, text=''.join(fo.readlines()))
    except:
        return web.Response(status=500, text=str([method, path]))

async def main(interface, port):
    proto  = web.Server(handler)
    server = await loop.create_server(proto, interface, port)

loop = asyncio.get_event_loop()
asyncio.Task(main('0.0.0.0', 8080))

# enter service loop
try:
    loop.run_forever()
except KeyboardInterrupt:
    print('')
    print('STATUS: Exiting ...')
    loop.close()
    exit(0)
```

```
$ curl localhost:8080/etc/hostname
thera
```

Cairo

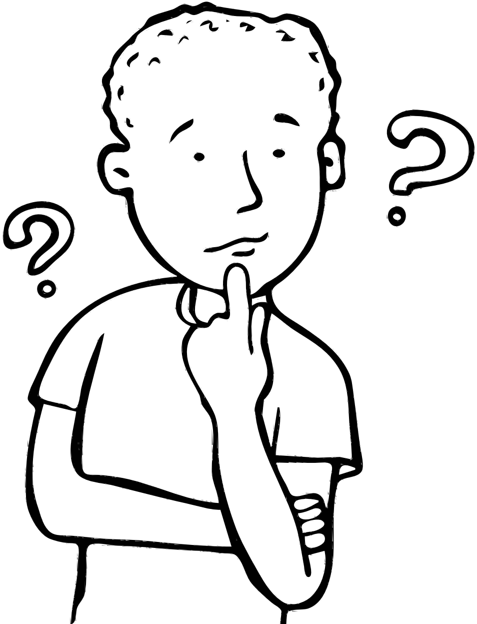
```
import cairo

mm = lambda value: float(value)/25.4*72
width, height = mm(60), mm(100)

surface = cairo.PDFSurface('cairoexample.pdf', width, height)
c = cairo.Context(surface)

c.move_to(0, 0)
c.line_to(width, height)
c.set_line_width(7.8)
c.set_source_rgb(0.8, 0.8, 1)
c.stroke()

c.move_to(0,0)
c.line_to(width,0)
c.line_to(0,height)
c.close_path()
c.set_source_rgba(0, 1, 0, 0.5)
c.fill()
```



Other Resources

- This material: `https://github.com/aslakjohansen/enigma-python-intro`
- *Think Python 2nd Edition* by Allen B. Downey (free PDF)
  `https://greenteapress.com/wp/think-python-2e/`
- *Think Stats 2nd Edition* by Allen B. Downey (free PDF)
  `https://greenteapress.com/wp/think-stats-2e/`

Questions and Comments?



https://openclipart.org/detail/238687/boy-thinking-of-question