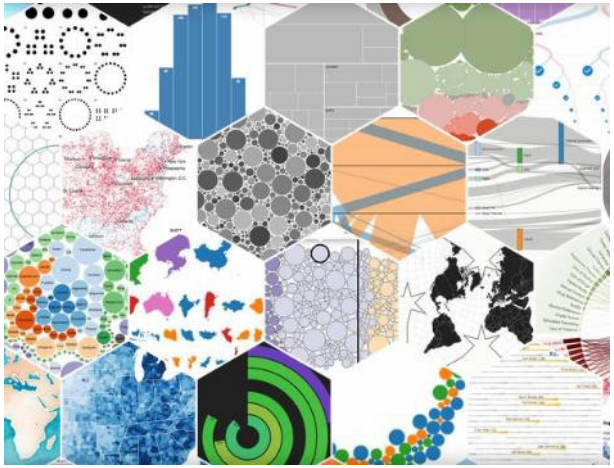
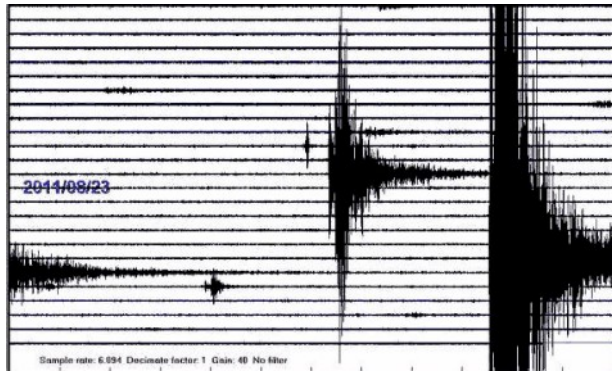
XVIII Escola de Verão
IAG/USP

Introdução a linguagem Python

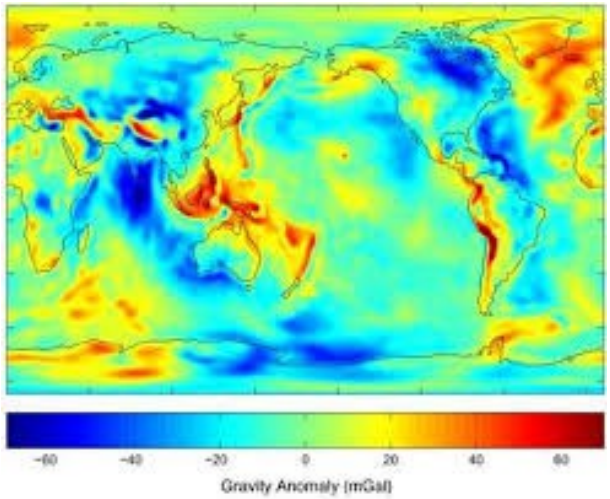
Marcelo Bianchi
Victor Sacek
Leonardo Uieda

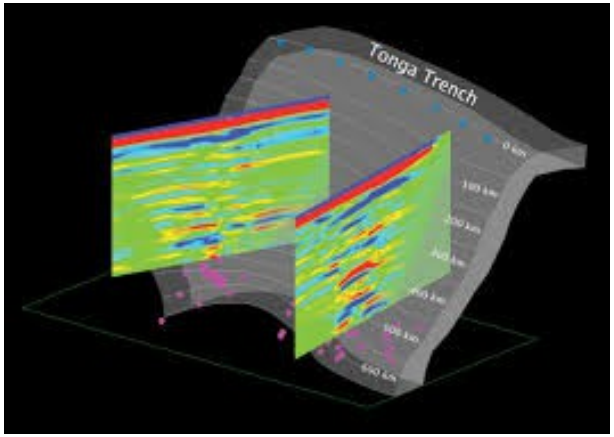
Jan/2016

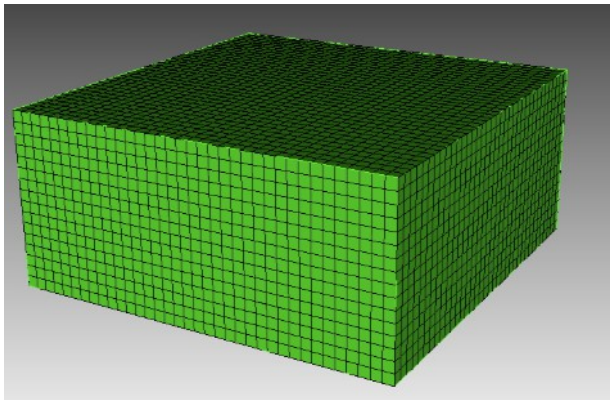


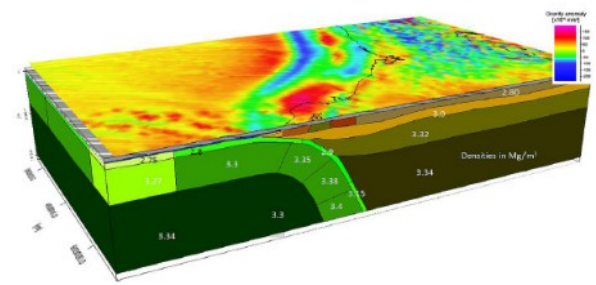

2011/08/23
Sample rate: 6.894 Decimate factor: 1 Gain: 40 No filter

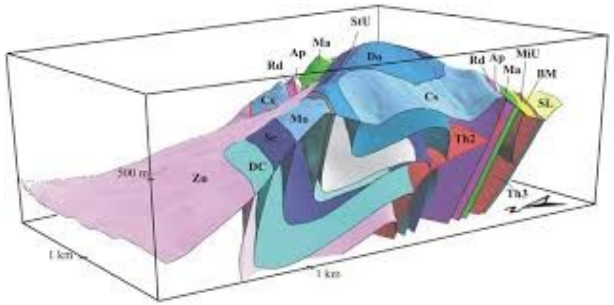

Statistics
organization
descriptive
scientific
collection
data
interpretation
statistical
inference
theory
experiments
mathematics
probability
discipline
population
prediction
science
called
mathematical
parameters
provides
also
word
wide
analysis
samples
total
design
pertaining
referring
branch
consider
information


-60
-40
-20
0
20
40
60
Gravity Anomaly (mGal)

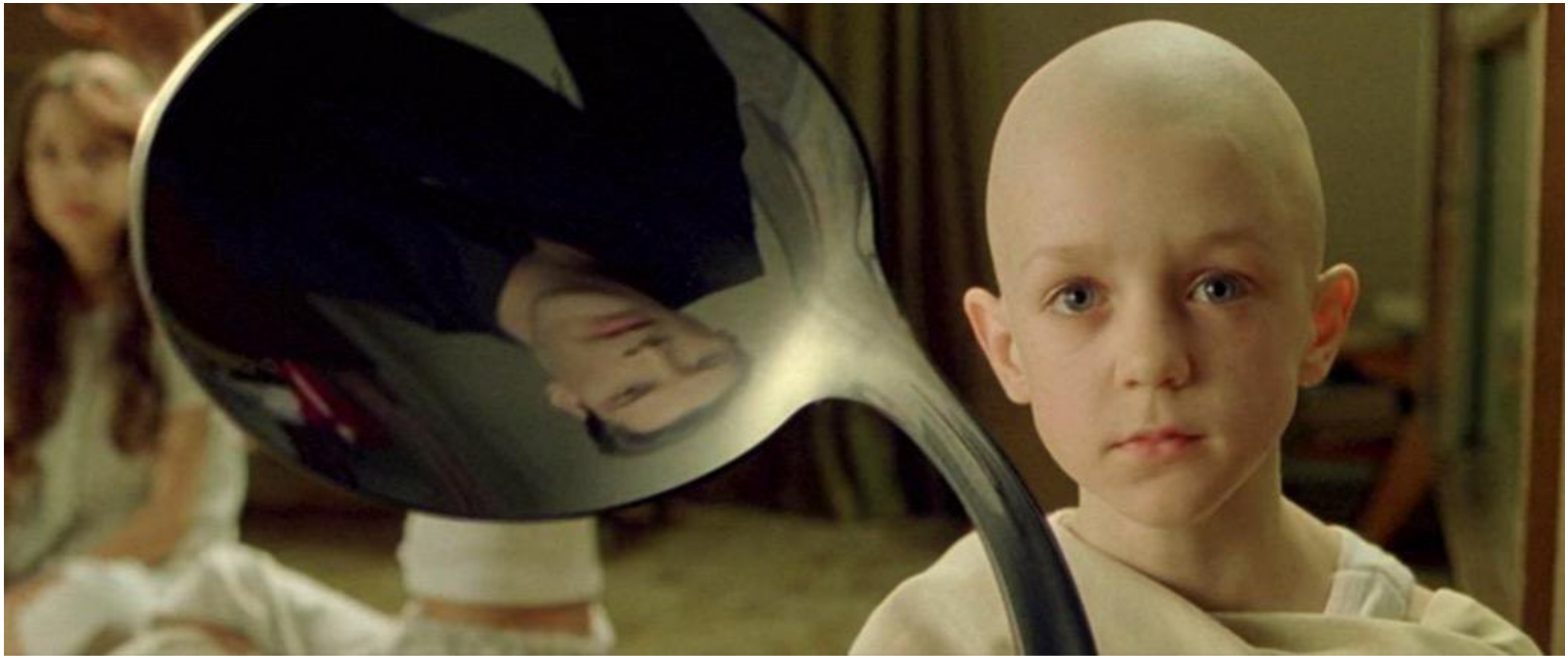

Tonga Trench

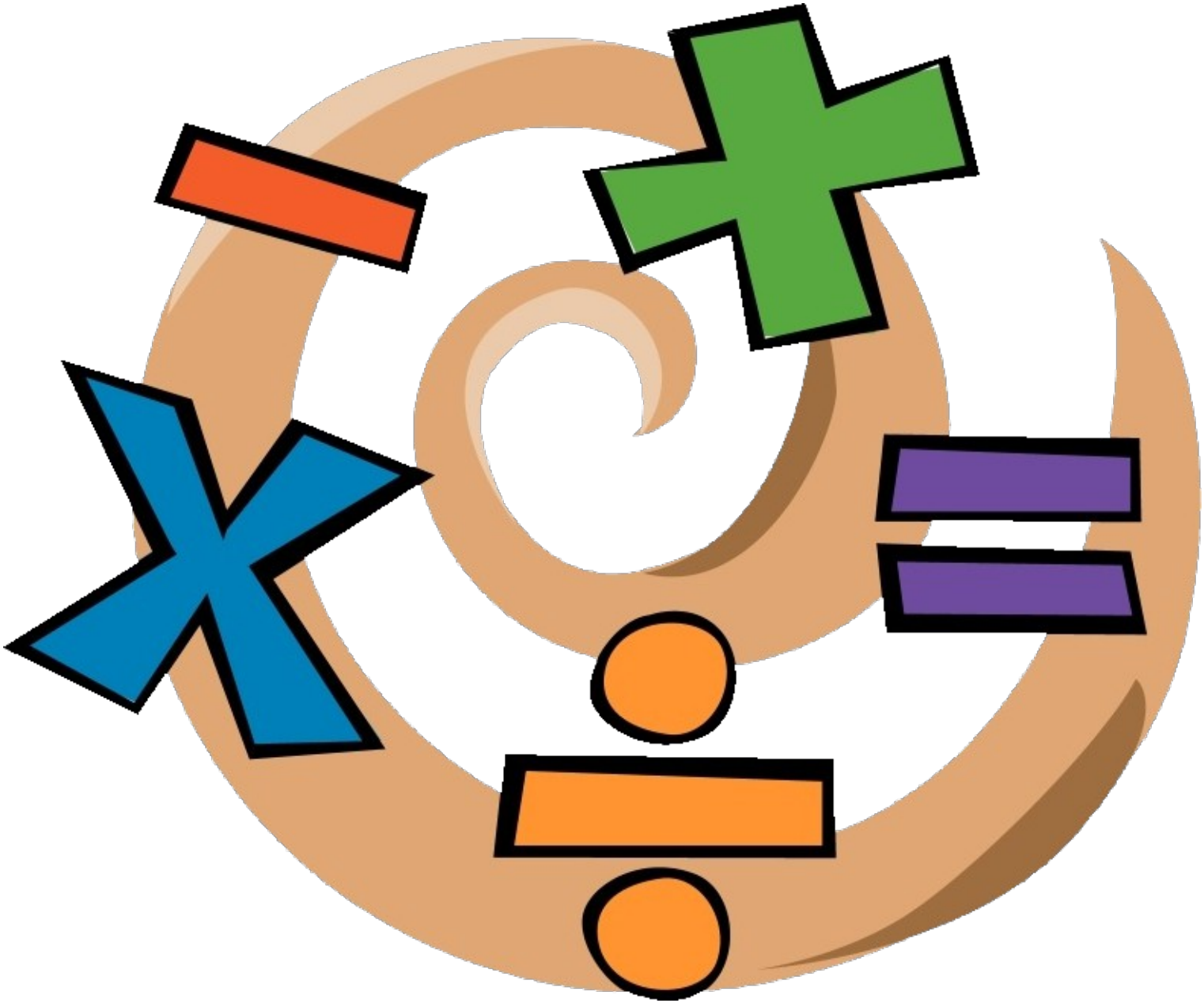


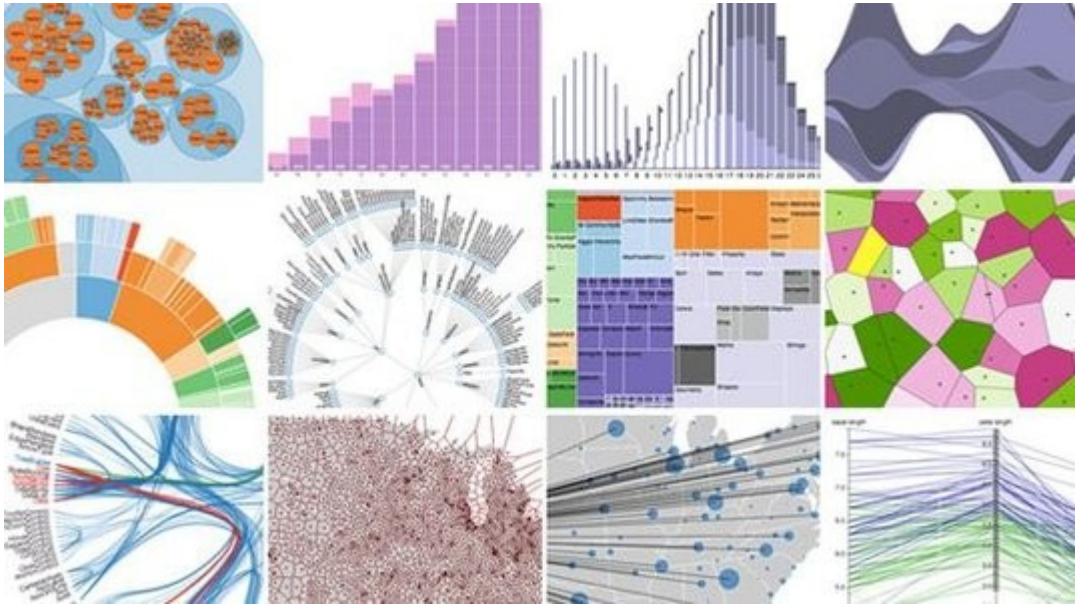

Densities in Mg/m³


StU
Ap
Ma
Rd
Do
Rd
Ap
MtU
Ma
BM
SL
Cs
Cs
Ms
Sc
Th2
DC
Za
500 m
Th3
1 km
1 km

```
11 '''
14 TIMEOUT = 60
15
16 class Error(object):
21
22 class LogAnaliser(object):
23
24     class LOGT(object):
27
28     def __init__(self, station, logfolder, network = "BL"):
29         if not os.path.isdir(logfolder):
30             try:
31                 os.mkdir(logfolder)
32             except OSError,e:
33                 raise Exception("Error making log folder !")
34
35         if not station:
36             raise Exception("Invalid station")
37
38         if not network:
39             raise Exception("Invalid network")
40
41         self._s = station
42         self._lf = logfolder
43         self._n = network
44
45     def _url(self, date, logtype):
46         url = "http://%s.bra-sis.net:%d/logs/" % (self._s, 5555 if self._s not in EXCEPTION_LIST else 80)
47
48         if logtype == self.LOGT.apollo:
49             url += date.strftime("%s_%%Y%%m%%d-001.log" % logtype)
50         else:
51             url += date.strftime("%s_%%Y%%m%%d-01.log" % logtype)
52
53         return url
54
55     def _fetch(self, url, filename):
56         '''
61
62         try:
63             logrequest = requests.get(url, timeout = TIMEOUT)
64         except requests.exceptions.Timeout:
65             return None
66
67         if logrequest.status_code != 200:
68             logrequest.close()
69             return None
70
71         ## All Good !
72         zipfile = gzip.open(filename,"w")
73         zipfile.write(logrequest.text.encode("utf-8"))
74         zipfile.close()
75
76         return gzip.open(filename, "r")
```







python™

Sobre o Python

- Linguagem criada por Guido van Rossum no final da década de 1980, e implementada inicialmente em 1989. É uma linguagem que suporta diferentes paradigmas sendo os principais: Programação orientada a objeto, imperativa, procedural ou funcional. Uma das suas principais características é ter tipos dinâmicos, gerenciamento de memória com coletor de lixo automático e uma extensiva biblioteca padrão !

python™

Do Terminal

```
mbianchi@a1200: ~
mbianchi@a1200:~$ python
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+1
2
>>>
```

De um arquivo

```
mbianchi@a1200: ~
mbianchi@a1200:~$ python 1and1.py
2
mbianchi@a1200:~$
```

```
1and1.py (~) - pluma
File  Edit  View  Search  Tools  Documents  Help
1and1.py
1 print 1+1
```

Façam em um terminal

```
$ python
>>> import this
```

```
In [1]: 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!

In [2]:
```

```
In [1]: 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!

In [2]:
```

```
In [1]: 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!

In [2]:
```

Simplicidade, Clareza & Consistência ao longo do programa

```
In [1]: 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!

In [2]:
```

Use as ferramentas da linguagem para construir um código complexo mas não complicado !

```
In [1]: 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!

In [2]:
```

Modularidade ao invés de if's, uso de namespaces e ampla biblioteca padrão

```
In [1]: 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!

In [2]:
```

Espaços e pontuação são parte da linguagem !

```
In [1]: 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!

In [2]:
```

Tudo é objeto, mesmo que você consiga programar em diferentes paradigmas

```
In [1]: 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!

In [2]:
```

Suporte a exceções e operações bem definidos mês sem tipos fixos

```
In [1]: 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!

In [2]:
```

Grande abundância de módulos, com uma linguagem extensível e dinâmica (Py 2.7 e 3.0), mas nem tudo, vai na biblioteca padrão.

```
In [1]: 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!

In [2]:
```

Simplicidade, sem perder complexidade !

```
In [1]: 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!

In [2]:
```

Encapsulamento de tarefas e informação auto contidas e reutilizáveis ! Mantendo a procedência !

IPython Notebook

```
mbianchi@a1200: ~
mbianchi@a1200:~$ ipython
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
Type "copyright", "credits" or "license" for more information.

IPython 2.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

O Python interativo (IPython) é um programa que mantém o Python rodando ao fundo criando um ambiente especial para o usuário interagir com o interpretador de uma forma exploratória

IP[y]:

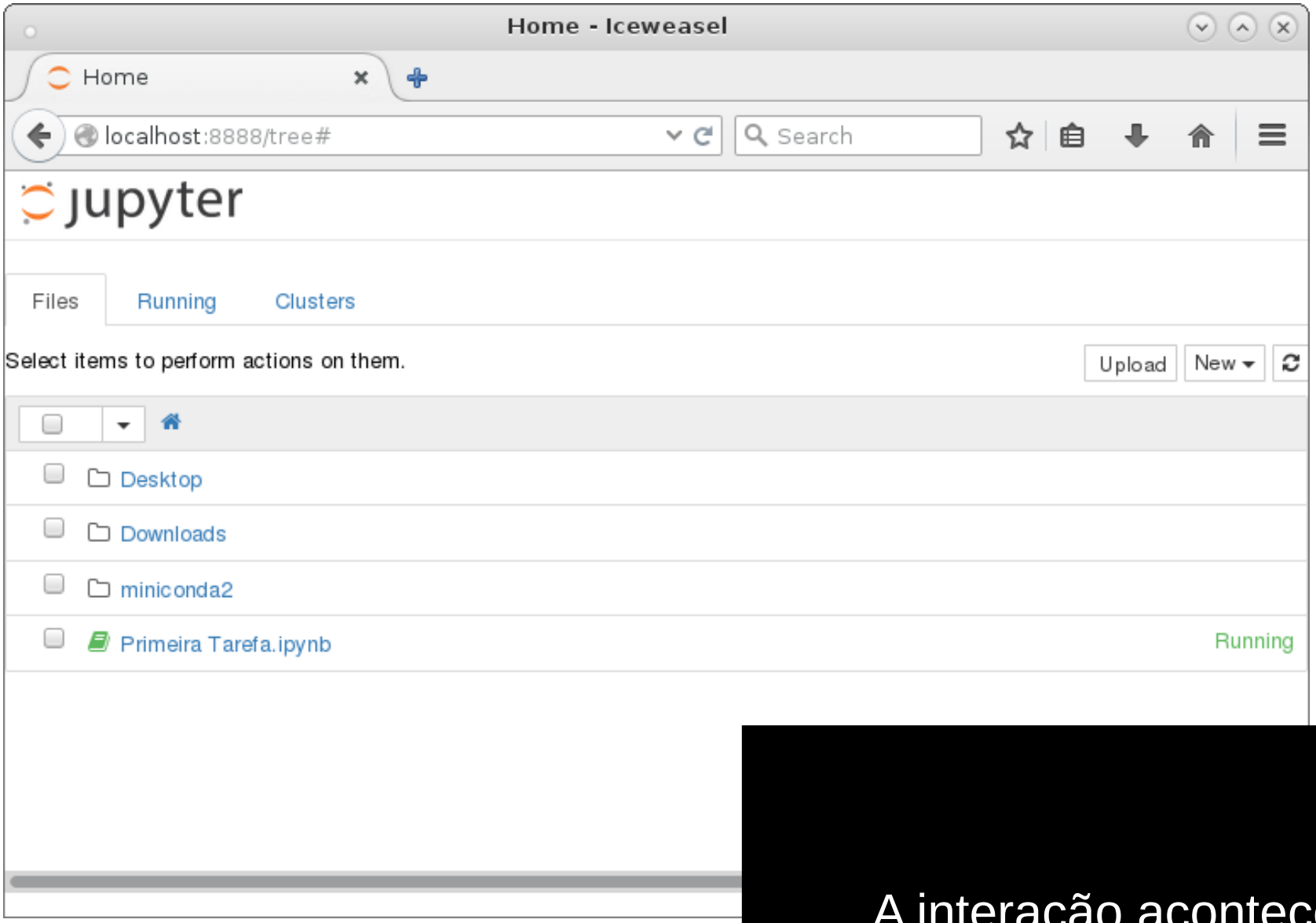
IPython

Interactive Computing

Jupyter Notebook

- Era parte do Python Interativo
- Hoje é um projeto para criar uma rica experiência em programação em conjunto com IPython e mesmo, com suporte a outras linguagens.
- Ele é iniciado como:

```
curso@CursoFatiando: ~
File  Edit  View  Search  Terminal  Help
curso@CursoFatiando:~$ ipython notebook
[W 15:02:38.725 NotebookApp] ipywidgets package not installed.  Widgets are unavailable.
[I 15:02:38.734 NotebookApp] Serving notebooks from local directory: /home/curso
[I 15:02:38.734 NotebookApp] 0 active kernels
[I 15:02:38.734 NotebookApp] The IPython Notebook is running at: http://localhost:8888/
[I 15:02:38.735 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```



A interação acontece via navegador

Untitled - Iceweasel

Untitled

localhost:8888/notebooks/DiaUm/Untitled.ipynb?kernel_name=python2

Search

jupyter Untitled

File Edit View Insert Cell Kernel Help Python 2

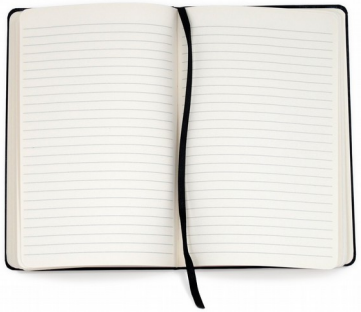
Code CellToolbar

```
In [1]: from __future__ import print_function, division

In [2]: print(1+1)

        2

In [ ]:
```

O notebook, é como se fosse um arquivo, com células de código, comentário e resultados.

Cada célula pode ser executada individualmente.

Agora Você



- Faça o programa “HelloWorld” de três maneiras diferentes:
  1) Utilizando o comando “python”;
  2) Abrindo um editor de texto, escrevendo as instruções no editor, salvando e executando o Python;
  3) Utilizando o ipython notebook;
- Utilize para isso a função ***print*** , como mostrado no exemplo anterior
- Se ficar em dúvida de como utilizar o print, use o comando help(print)

O Básico

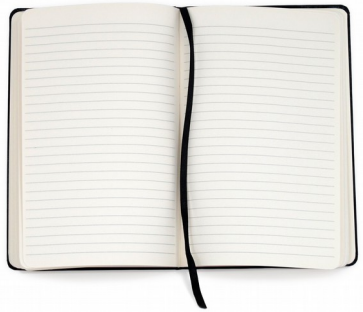
- Comentários são indicados por #, possível também colocar comentários com diversas linhas no python utilizar aspas triplas (”””) para criar uma string anônima que aceita \n dentro dela
- Tipos de variáveis são definidos dinamicamente;
- Indentação e pontuação definem os blocos;
- Os tipos básicos podem ser:
  - ***None*** (Tipo especial que representa o nada), ***int*** (inteiro), ***float*** (ponto flutuante), ***bool*** (boleano), **str** (texto), ***complex*** (complexo), ***list*** (lista), ***tuple*** (tupla) & ***dict*** (dicionário)
- Os principais laços são: **if**, **while**, **for**
- A função **print** imrpime as variáveis, **type** o seu tipo, e as funções **range** e **xrange** geram listas de números inteiros;
- Colchetes (**[...]**) indicam elementos em listas e tuplas;
- e o comando interno **help**() mostra a ajuda para todo objeto no PYTHON !

Python 2 vs Python 3

```
from __future__ import print_function, division
```

- A linguagem Python nos últimos anos vem migrando lentamente da versão 2 para a versão 3 e para auxiliar na migração existe o módulo future.
- É aconselhável sempre adicionar a linha acima nos seus programas de Python 2, para este já ficarem compatíveis com a sintaxe do Python 3 e facilitar a migração !

Agora é com você



- Abra o IPython notebook e complete o notebook “PrimeirosPassos.pynb” dentro da pasta EscolaVerao/DiaUm
  - No notebook tem mais explicações, intercaladas com células de treino, faça cada uma delas !
  - Para executar uma célula, você deve selecionar a célula e em seguida, pressionar Ctrl+Enter, é importante notar que algumas células dependem de células anteriores já que UM notebook é um programa sequencial !