



**INSTITUTO POLITÉCNICO DE  
BEJA**

**Processamento de estatísticas de  
candidaturas do ensino superior  
de primeira fase**

**3/12/2013**



**Marco Filipe do Carmo  
Sacristão  
Aluno nº 11907**



**Jorge Daniel Margarido  
Batista  
Aluno nº 6307**

# Conteúdo

Introdução.....	2
Teoria .....	3
Parte Experimental .....	4
Realização Experimental. ....	4
Sistema Experimental .....	5
Resultados Experimentais .....	15
Conclusão .....	16
Bibliografia.....	17

# Introdução

## Contextualização do trabalho

A linguagem de programação Python é na sua essência uma linguagem de programação de **uso-geral**<sup>1</sup> e de **alto nível**<sup>2</sup>, sendo também **interpretada**<sup>3</sup> e **imperativa**<sup>4</sup>, assim como **orientada a objetos**<sup>5</sup>.

1. Uma linguagem de programação de uso-geral é uma linguagem de programação cuja aplicação está desenhada para poder vir a ser aplicada em vários domínios.
2. Uma linguagem de programação de alto nível é constituída um nível de abstração que permite com que a linguagem em questão não tenha de estar dependente de elementos relativos ao controlo de gestão da máquina (baixo nível), por exemplo, em Python não é necessário uma gestão minuciosa de memória, que em conjunto com outros fatores a torna uma linguagem de alto nível.
3. A linguagem interpretada é normalmente uma linguagem onde o código fonte é diretamente executada pelo interpretador pelo seu código fonte sem passar pelo processo de compilação. No entanto, existem também linguagens de programação que podem ser consideradas interpretadas e também compiladas.
4. Nas linguagens imperativas, a programação imperativa é um paradigma de programação que no qual está implícito a computação através de ações, enunciados ou comandos que se alteram de estado no decorrer do programa em questão.
5. A programação orientada a objetos é também um paradigma da programação, mas não só, pois poder ser aplicada à análise a projeção.

Esta linguagem de programação foi lançada por **Guido van Rossum** em 1991, mas na actualidade possui um modelo de desenvolvimento comunitário, o que significa que apesar da existência uma organização sem fins lucrativos que a gere (**Python Software Foundation**), a sua composição é quase na sua totalidade submetida por membros da comunidade.

## Descrição dos objetivos realizados

Relativamente aos objetivos propostos no enunciado deste projeto e a sua realização, queremos acreditar que todos os objetivos terão sido completados na sua totalidade, e achamos também, que existem alguns fatores que apesar de não nos serem requisitados, decidimos que seriam de boa prática a sua implementação neste projeto de modo a melhorar a sua estruturação e desempenho. Dentro deste contexto, definiu-se que foi decidido por nossa parte a utilização da biblioteca de mapeamento objeto-relacional SQL **SQLAlchemy**, assim como um conjunto de asserções no próprio código que verificam a correta definição de fatores cruciais à total execução do projeto em si, evitando através de avisos qualquer ambiguidade inconsciente que possa vir a existir (Ex: Tentativa de leitura de um ficheiro inexistente).

## Estrutura do Relatório

O nosso relatório encontra-se dividido em quatro partes essenciais:

- **Teoria**
- **Parte Experimental**
  - *Realização Experimental*  
Dados sobre a linguagem de programação utilizada, ambiente de desenvolvimento utilizado para execução do projeto, sistema operativo utilizado e o hardware onde foi desenvolvido.
  - *Sistema Experimental*  
Descrição detalhada da estruturação do código realizado para execução do projeto.
  - *Resultados Experimentais*  
Protocolo experimental usado para obtenção dos resultados assim como uma breve apresentação e discussão dos dados obtidos.
- **Conclusão**
- **Bibliografia**

# Teoria

Começando pelo módulo responsável pela leitura do ficheiro do qual são obtidos os dados que nos foram fornecidos, o **xlrd** é uma biblioteca que permite a extração de dados de uma folha Microsoft Excel.

Passando assim para a base de dados para onde é enviada toda a informação pertinente na folha Excel fornecida, o **SQLAlchemy** é uma biblioteca de mapeamento objeto-relacional SQL, cujo principal objetivo é fornecer ao programador uma estruturação de gestão sobre uma base de dados SQL utilizando modelos de programação orientada a objeto, ao invés da normal utilização da sintaxe SQL.

Após a leitura dos dados da folha Excel ao utilizar **xlrd**, e a sua conversão para uma base de dados SQL utilizando **SQLAlchemy**, os dados relativos às estatísticas são extraídos através de uma série de algoritmos relativos à estatística requisitada sendo mais tarde armazenados por outra biblioteca designada por **CSV** (Comma Separated Values) a qual permite que um conjunto de ficheiros seja armazenado num ficheiro separado apenas por vírgulas, o que nos permite então agora uma rápida extração dos resultados das estatísticas realizadas sobre os dados de modo a poder mais tarde realizar gráficos de barras e também gráficos de pizza para a então final análise dos dados pelo utilizador final.

Para a realização dos gráficos em questão é utilizada uma outra biblioteca com o nome **matplotlib** sendo que esta permite aos programadores a realização de um leque de desenho de funções gráficas em duas dimensões.

Como etapa final deste projeto, foi-nos requisitado a realização de uma interface gráfica onde o utilizador final poderá realizar todas as funções de transformação de dados e visualização dos seus resultados, para este efeito, decidimos enquanto grupo que iríamos utilizar a biblioteca **Tkinter** que não é nada mais do que um standard GUI (Graphical User Interface) que funciona sobre uma camada orientada a objectos “em cima” de **Tcl/Tk** em que **Tcl** é uma linguagem interpretada, muito portátil e extensível, criada em 1988 por John Ousterhout e **Tk** é um conjunto de ferramentas de GUI para **Tcl**.

# Parte Experimental

## Realização Experimental.

Como já havia sido definido anteriormente, a linguagem de programação utilizada para a execução deste projeto foi Python (versão 2.7.3).

O ambiente de desenvolvimento sob o qual este projeto foi executado foi:

- **Sublime Text 3**  
Para a realização e edição do código Python do projecto.
- **Mercurial**  
Sistema de gestão e controlo de versões para código fonte distribuído.

Relativamente ao sistema operativo utilizado, todo o código e gestão do mesmo foi feito apenas utilizando uma distribuição Linux com o nome **Debian** (Versão Wheezy), o **hardware** sob o qual o sistema operativo corria era uma máquina virtual criada através do software **VirtualBox**, com uma alocação de memória de apenas um 1GB.

## Sistema Experimental

No decorrer desta secção que será explicado grande parte do código, de modo a não criar um grande fluxo de texto no documento, será apenas feita uma referência a imagens e ao ficheiro Python em questão seguido de uma breve explicação de algum código que eventualmente necessite de alguma especificação extra que não seja de normal perceção.

data\_manager.py

Como foi definido na projeção, teria de existir um ficheiro responsável pela leitura da folha Excel, e após a sua leitura (do respetivo intervalo com dados a serem extraídos que pode ser definido como variável global) assim como nome do ficheiro a ser lido. Após a leitura dos dados para um formato mais fácil de manusear com a linguagem, criámos o modelo de relações de SQLAlchemy permitindo mais tarde ser feita a seleção de colunas necessárias e preenchemos então a base de dados e tabela criada com os dados lidos.

```
19 from sqlalchemy import Column, Integer, String, Float, create_engine
20
21 # File's name to open using XLRD
22 FILE_NAME = "cna131fresultados.xls"
23 # Sheet number that contains the information
24 SHEET_NUMBER = 0
25
26 # Open FILE_NAME forcing utf-8 encoding and set current sheet as 0
27 try:
28     OPEN_FILE = xlrd.open_workbook(FILE_NAME, encoding_override="utf-8")
29     SHEET = OPEN_FILE.sheet_by_index(SHEET_NUMBER)
30     """
31     'Row offsets' are used to know from what interval we should start
32     extracting data from, this way we can exclude any rows that do not contain
33     information that should be extracted. This way we can guarantee that
34     changes to the row structure can be easily altered below to read
35     from a different interval.
36     """
37     ROW_INTERVAL_START = 3
38     ROW_INTERVAL_END = SHEET.nrows - 2 #SHEET.nrows = Total rows in file.
39 except:
40     print FILE_NAME + " not found in current directory!"
41
42
43 # SQLAlchemy defining declarative base class, connection and debugging.
44 try:
45     # populate_database() needs to be ran in order to Create Database.
46     BASE = declarative_base()
47     DATABASE = create_engine('sqlite:///database.db', echo=True)
48     DB_SESSION = scoped_session(sessionmaker())
49     DB_SESSION.configure(bind=DATABASE, autoflush=False, expire_on_commit=False)
```

Foi criado também uma função que nos permite verificar através do nome da instituição qual o distrito em que a mesma se encontra “check\_district()” e recebe como parâmetro “institution\_name”

```
def check_district(institution_name):
    # List of districts that exist in Portugal
    available_districts = ['Lisboa', 'Porto', u'Setúbal', 'Braga', 'Aveiro',
                           'Leiria', u'Santarém', 'Faro', 'Coimbra', 'Viseu', 'Madeira', u'Açores',
                           'Viana do Castelo', 'Vila Real', 'Castelo Branco', u'Évora', 'Guarda', 'Beja',
                           u'Bragança', 'Portalegre']

    ...

    Checks the Institution name for any available District, if it finds one it
    return the district that was found in the name and set it as the
    Institution's district
    ...

    for i in range(0, len(available_districts)):
        if(available_districts[i] in institution_name):
            return available_districts[i]

    # List of regions to be associated with a district that exists in Portugal
    unset_districts = ['Algarve', 'Beira Interior', 'Minho',
                       u'Trás-os-Montes e Alto Douro', u'Cávado e do Ave', 'Tomar',
                       u'Náutica Infante D. Henrique', 'Estoril']

    ...

    After checking the Institution's name and no available district is found
    within, we should convert the Region that we find in the name to an
    available district
    ...

    for i in range(0, len(unset_districts)):
        if(unset_districts[i] in institution_name):
            if(i==0): # Region Algarve found
                return available_districts[7] # Return Faro
            if(i==1): # Region Beira Interior(Covilhã) found
                return available_districts[14] # Return Castelo Branco
            if(i==2): # Region Minho found
                return available_districts[3] # Return Braga
            if(i==3): # Region Trás-os-Montes e Alto Douro found
                return available_districts[13] # Return Vila Real
            if(i==4): # Region Cávado e do Ave found
                return available_districts[3] # Return Braga
            if(i==5): # Region Tomar found
                return available_districts[6] # Return Santarém
            if(i==6): # Region Paço de Arcos -> Oeiras
                return available_districts[0] # Return Lisboa
            if(i==7): # Region Estoril -> Cascais
                return available_districts[0] # Return Lisboa

    return "Unknown District"
```

Se não for possível detetar o nome do distrito no nome da instituição, a função irá então transformar a região presente no nome da instituição para o respetivo distrito. Ex: *Universidade do Algarve*, o distrito seria *Faro*.

statistics\_handler.py

Este ficheiro primeiro garante que é possível estabelecer uma ligação à base de dados, só após esta garantia será possível através deste módulo fazer as estatísticas requeridas no relatório.

Fica abaixo uma tabela com a estatística requisitada e a função responsável por essa estatística.

<b>Número de alunos colocados por instituição</b>	students_placed_by_institution()
<b>Número de alunos colocados por distrito</b>	students_placed_by_district()
<b>Permilagem de alunos colocados por distrito</b>	per_mil_students_placed_by_district()
<b>Percentagem de alunos colocados por instituição em relação a todos os alunos colocados</b>	get_all_placed() & percentage_all_students_placed_by_institution()
<b>Número de vagas por colocar por instituição</b>	openings_remaining_by_district()
<b>Número de vagas por color por distrito</b>	openings_remaining_by_institution()



Seguem-se as imagens referentes ao código das funções acima referidas.

```
# Number of students placed by Institution
def students_placed_by_institution():

    # Name of the file to be handled by the current function
    file_name = "students_placed_by_institution.csv"

    # CSV File to contain statistics current function
    try:
        csvfile = csv.writer(open(file_name,"wb"),
                               quoting=csv.QUOTE_ALL)
        print "Successfully opened file: " + str(file_name)
    except:
        print "Failed to open CSV file: " + str(file_name) + "!"

    # Run a query Selecting all elements
    query = RESULTS.select()
    run_query = query.execute()

    ...

    We use this variable to only count all the placed students
    once per institute
    ...

    last_institution_code = 0

    # For each row in the previous query
    for column in run_query:
        new_institution_code = column[1] # column[1] being InstituteCode
        result_list = [] # Creates a list to which data will be appended
        if last_institution_code != new_institution_code:
            result_list.append(column[3].encode('utf-8')) # Apnd InstitutionName
            ...

            The code below selects from the query all the entries that
            match the given InstitutionCode and adds the students
            that got placed in each row value until
            another InstitutionCode is detected
            ...

            get_all_institution_by_id = RESULTS.select(
                RESULTS.c.InstitutionCode.like(int(column[1])))
            getocurrencies = get_all_institution_by_id.execute()
            placed_students = 0
            for x in getocurrencies:
                placed_students = placed_students + int(x[7]) # No.Placed in row
            result_list.append(placed_students) # Appends total placed students
            csvfile.writerow(result_list)
        last_institution_code = column[1]
```

```

def students_placed_by_district():
    # Name of the file to be handled by the current function
    file_name = "students_placed_by_district.csv"

    # CSV File to contain statistics current function
    try:
        csvfile = csv.writer(open(file_name,"wb"),
                               quoting=csv.QUOTE_ALL)
        print "Successfully opened file: " + str(file_name)
    except:
        print "Failed to open CSV file: " + str(file_name) + "!"

    ...

    Run a query Selecting all elements and ordering it by District Ascending
    ...

    query = RESULTS.select()
    order_by_asc = query.order_by(asc(RESULTS.c.District))
    run_query = order_by_asc.execute()

    ...

    We use this variable to only count all the placed students
    once per institute
    ...

    last_district = ""

    # For each row in the previous query
    for column in run_query:
        ...

        column[10] being District
        ...

        new_institution_code = column[10].encode('utf-8')
        result_list = [] # Creates no list to which data will be appended
        if last_district != new_institution_code:
            ...

            Append InstitutionName
            ...

            result_list.append(column[10].encode('utf-8'))
            ...

            The code below selects from the query all the entries that
            match the given District and adds the students
            that got placed in each row value until
            another District is detected
            ...

            get_all_intitution_by_id = RESULTS.select(
                RESULTS.c.District.like(column[10]))
            getocurrencies = get_all_intitution_by_id.execute()

            placed_students = 0
            for x in getocurrencies:
                placed_students = placed_students + int(x[7]) # No.Placed in row
            result_list.append(placed_students) # Appends total placed students
            csvfile.writerow(result_list)
            last_district = column[10].encode('utf-8')

```

```

def per_mil_students_placed_by_district():
    # Name of the file to be handled by the current function
    file_name = "per_mil_students_placed_by_district.csv"

    # CSV File to contain statistics current function
    try:
        csvfile = csv.writer(open(file_name,"wb"),
                               quoting=csv.QUOTE_ALL)
        print "Successfully opened file: " + str(file_name)
    except:
        print "Failed to open CSV file: " + str(file_name) + "!"

    ...

    Run a query Selecting all elements and ordering it by District Ascending
    ...

    query = RESULTS.select()
    order_by_asc = query.order_by(asc(RESULTS.c.District))
    run_query = order_by_asc.execute()

    ...

    We use this variable to only count all the placed students
    once per institute
    ...

    last_district = ""

    # For each row in the previous query
    for column in run_query:
        ...

        column[10] being District
        ...

        new_institution_code = column[10].encode('utf-8')
        result_list = [] # Creates no list to which data will be appended
        if last_district != new_institution_code:
            ...

            Append InstitutionName
            ...

            result_list.append(column[10].encode('utf-8'))
            ...

            The code below selects from the query all the entries that
            match the given District and adds the students
            that got placed in each row value until
            another District is detected
            ...

            get_all_intitution_by_id = RESULTS.select(
                RESULTS.c.District.like(column[10]))
            getocurrencies = get_all_intitution_by_id.execute()

            placed_students = 0
            for x in getocurrencies:
                placed_students = placed_students + int(x[7]) # No.Placed in row
            ...

            Appends total placed students * 0.001 (Per mil)
            Formatting to float only 3 decimals
            ...

            result_list.append("{0:.3f}".format(placed_students*0.001))
            csvfile.writerow(result_list)
            last_district = column[10].encode('utf-8')

```

```
def get_all_placed():
    # Run a query Selecting all elements
    query = RESULTS.select()
    run_query = query.execute()
    total_placed = 0.0
    for i in run_query:
        total_placed = total_placed + i[7]

    return total_placed
```

```
def percentage_all_students_placed_by_institution():
    # Name of the file to be handled by the current function
    file_name = "percentage_all_students_placed_by_institution.csv"

    # CSV File to contain statistics current function
    try:
        csvfile = csv.writer(open(file_name,"wb"),
                               quoting=csv.QUOTE_ALL)
        print "Successfully opened file: " + str(file_name)
    except:
        print "Failed to open CSV file: " + str(file_name) + "!"

    # Run a query Selecting all elements
    query = RESULTS.select()
    run_query = query.execute()

    ...

    We use this variable to only count all the placed students
    once per institute
    ...

    last_institution_code = 0

    # For each row in the previous query
    for column in run_query:
        new_institution_code = column[1] # column[1] being InstituteCode
        result_list = [] # Creates a list to which data will be appended
        if last_institution_code != new_institution_code:
            result_list.append(column[3].encode('utf-8')) # Apnd InstitutionName
            ...

            The code below selects from the query all the entries that
            match the given InstitutionCode and adds the students
            that got placed in each row value until
            another InstitutionCode is detected
            ...

            get_all_institution_by_id = RESULTS.select(
                RESULTS.c.InstitutionCode.like(int(column[1])))
            getocurrencies = get_all_institution_by_id.execute()
            placed_students = 0
            for x in getocurrencies:
                placed_students = placed_students + int(x[7]) # No.Placed in row
            ...

            Appends total placed students * 0.01 (Percentage)
            Formatting to float only 2 decimals
            ...

            # Gets all placed students with get_all_placed()
            total_placed = get_all_placed()
            # Calculates the percentage of placed students
            percentage_placed = (placed_students*100)/total_placed
            result_list.append("{0:.2f}".format(percentage_placed))
            csvfile.writerow(result_list)
            last_institution_code = column[1]
```

```

def openings_remaining_by_district():
    # Name of the file to be handled by the current function
    file_name = "openings_remaining_by_district.csv"

    # CSV File to contain statistics current function
    try:
        csvfile = csv.writer(open(file_name, "wb"),
                               quoting=csv.QUOTE_ALL)
        print "Successfully opened file: " + str(file_name)
    except:
        print "Failed to open CSV file: " + str(file_name) + "!"

    ...

    Run a query Selecting all elements and ordering it by District Ascending
    ...

    query = RESULTS.select()
    order_by_asc = query.order_by(asc(RESULTS.c.District))
    run_query = order_by_asc.execute()

    ...

    We use this variable to only count all the openings remaining
    once per District
    ...

    last_district = ""

    # For each row in the previous query
    for column in run_query:
        ...

        column[10] being District
        ...

        new_institution_code = column[10].encode('utf-8')
        result_list = [] # Creates no list to which data will be appended
        if last_district != new_institution_code:
            ...

            Append InstitutionName
            ...

            result_list.append(column[10].encode('utf-8'))
            ...

            The code below selects from the query all the entries that
            match the given District and adds the openings
            remaining in each row value until
            another District is detected
            ...

            get_all_intitution_by_id = RESULTS.select(
                RESULTS.c.District.like(column[10]))
            getocurrencies = get_all_intitution_by_id.execute()

            remaining_openings = 0
            for x in getocurrencies:
                # No.Openings Remaining in row
                remaining_openings = remaining_openings + int(x[9])
            # Appends total remaining openings
            result_list.append(remaining_openings)
            csvfile.writerow(result_list)
            last_district = column[10].encode('utf-8')

```

```

def openings_remaining_by_institution():

    # Name of the file to be handled by the current function
    file_name = "openings_remaining_by_institution.csv"

    # CSV File to contain statistics current function
    try:
        csvfile = csv.writer(open(file_name, "wb"),
                               quoting=csv.QUOTE_ALL)
        print "Successfully opened file: " + str(file_name)
    except:
        print "Failed to open CSV file: " + str(file_name) + "!"

    # Run a query Selecting all elements
    query = RESULTS.select()
    run_query = query.execute()

    ...

    We use this variable to only count all the openings remaining
    once per institute
    ...

    last_institution_code = 0

    # For each row in the previous query
    for column in run_query:
        new_institution_code = column[1] # column[1] being InstituteCode
        result_list = [] # Creates a list to which data will be appended
        if last_institution_code != new_institution_code:
            result_list.append(column[3].encode('utf-8')) # Append InstitutionName
            ...

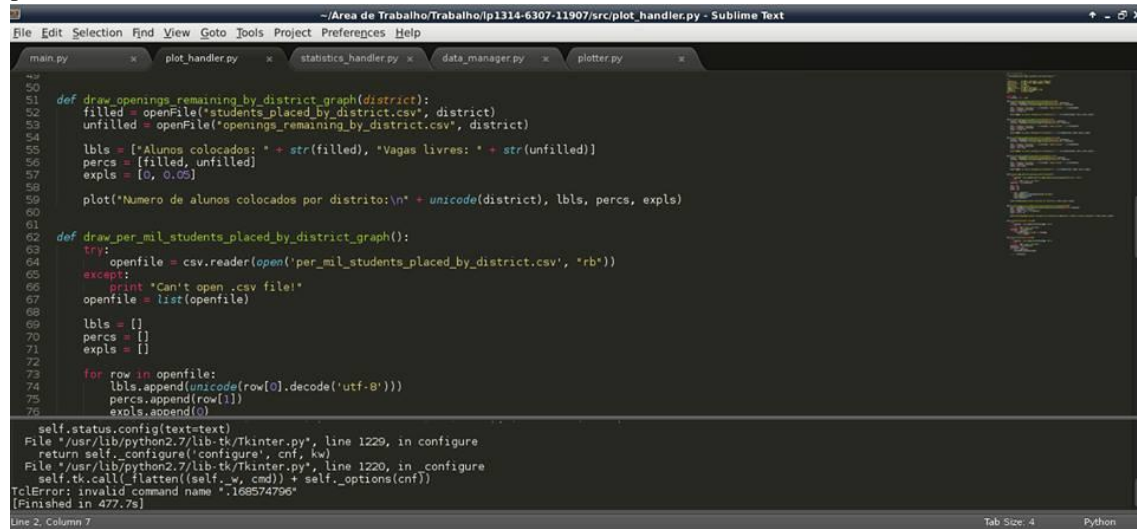
            The code below selects from the query all the entries that
            match the given InstitutionCode and adds the openings remaining
            in each row value until
            another InstitutionCode is detected
            ...

            get_all_institution_by_id = RESULTS.select(
                RESULTS.c.InstitutionCode.like(int(column[1])))
            getocurrencies = get_all_institution_by_id.execute()
            openings_remaining = 0
            for x in getocurrencies:
                # No.Placed in row
                openings_remaining = openings_remaining + int(x[9])
            # Appends total of openings remaining
            result_list.append(openings_remaining)
            csvfile.writerow(result_list)
            last_institution_code = column[1]

```

## plot\_handler.py

É nesta função onde usamos os ficheiros .CSV gerados para preencher graficamente os requisitos feitos pelo utilizador final ao utilizar a interface criada.

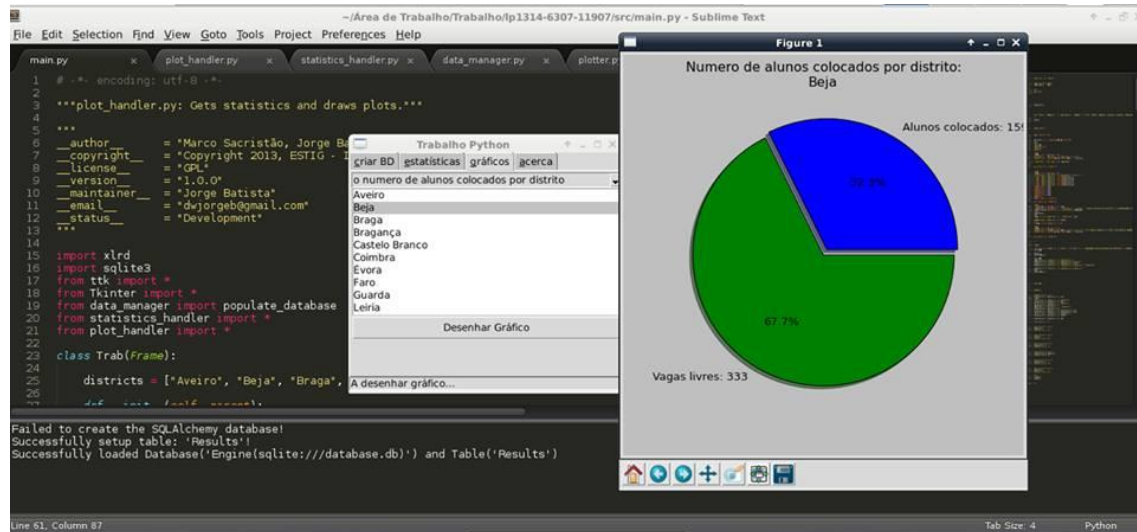


```
50
51 def draw_openings_remaining_by_district_graph(district):
52     filled = openFile('students_placed_by_district.csv', district)
53     unfilled = openFile('openings_remaining_by_district.csv', district)
54
55     lbls = ["Alunos colocados: " + str(filled), "Vagas livres: " + str(unfilled)]
56     percs = [filled, unfilled]
57     expls = [0, 0.05]
58
59     plot("Número de alunos colocados por distrito:\n" + unicode(district), lbls, percs, expls)
60
61
62 def draw_per_mil_students_placed_by_district_graph():
63     try:
64         openfile = csv.reader(open('per_mil_students_placed_by_district.csv', "rb"))
65     except:
66         print "Can't open .csv file!"
67         openfile = list(openfile)
68
69     lbls = []
70     percs = []
71     expls = []
72
73     for row in openfile:
74         lbls.append(unicode(row[0].decode('utf-8')))
75         percs.append(row[1])
76         expls.append(0)
77
78     self.status.config(text=text)
79
80 File "/usr/lib/python2.7/lib-tk/Tkinter.py", line 1229, in configure
81     return self._configure('configure', cnf, kw)
82 File "/usr/lib/python2.7/lib-tk/Tkinter.py", line 1220, in _configure
83     self.tk.call(_flatten((self._w, cmd)) + self._options(cnf))
84 TclError: invalid command name ".168574796"
85 [Finished in 477.7s]
```

## Resultados Experimentais

### Interface gráfica

Posto isto passamos então á parte gráfica da aplicação que foi criada através da biblioteca TkInter, onde podemos observar os menus e interfaces responsáveis pela execução das estatísticas e o bom funcionamento dos mesmos.





# Conclusão

Para concluir este relatório gostaríamos de evidenciar que ao visualizar o nosso repositório de código é de fácil interpretação que o trabalho foi realizado em grande parte num curto espaço de tempo, aumentando assim o desafio a nível pessoal para ambos os membros do grupo, dos quais **Marco Sacristão** ficou responsável pela (re)estruturação de todo o código base deste projeto, o qual viria a ser interpretado ao mesmo tempo por **Jorge Batista** de modo a ser possível a ligação entres os diferentes módulos do projeto a virem mais tarde a ser implementados na interface final do mesmo originando assim o produto final que aqui apresentamos.

Como ambos os membros do grupo estariam pré-formatados a outro tipo de sintaxe até este ponto, numa primeira impressão a sintaxe bastante “original” de Python relativamente ao uso de indentações e não seguindo o modelo de sintaxe “C-Like” como estaríamos habituados terá oferecido alguma resistência ao início da execução, mas foi rapidamente ultrapassada o que nos permitiu uma maior fluidez realização do código.

Introduzir o fator SQLAlchemy ao nosso projeto devido à aprendizagem necessária para a utilização do mesmo, pensamos que terá sido o maior obstáculo que encontrámos, não esquecendo algumas dificuldades que enfrentámos devido a “*encodings*” e formatações a ser utilizadas na passagem de dados, principalmente na criação de alguns gráficos utilizando o *matplotlib*, processo este que foi dificultado por estas passagens de dados com codificações diferentes, mas mesmo assim foi algo que resolvemos a nosso ver com bastante prontidão.

O desenvolvimento da interface gráfica utilizando Tkinter foi bastante fluído e não ofereceu grande dificuldade.

A nosso ver, a linguagem programação Python é uma das melhores linguagens de *scripting* da atualidade, o que pode ser comprovado pela sua grande adesão nos últimos anos, as suas convenções também elas são de rápida aprendizagem e fácil implementação dentro do contexto de estrutura do código. Ao realizar este projeto ambos os membros do grupo ganharam um maior afeto por esta linguagem, e foi desenvolvido um grande conjunto de novas capacidades dentro da mesma, aumentando assim o nosso nível de integração com a mesma.

# Bibliografia

- [Python: What is the common header format? - Stack Overflow](http://stackoverflow.com/questions/1523427/python-what-is-the-common-header-format)  
(<http://stackoverflow.com/questions/1523427/python-what-is-the-common-header-format>)
- [Epydoc Fields](http://epydoc.sourceforge.net/manual-fields.html#module-metadata-variables)  
(<http://epydoc.sourceforge.net/manual-fields.html#module-metadata-variables>)
- [Code Like a Pythonista: Idiomatic Python](http://python.net/%7Egoodger/projects/pycon/2007/idiomatic/handout.html)  
(<http://python.net/%7Egoodger/projects/pycon/2007/idiomatic/handout.html>)
- [Calling a function in a separate file in Python - Stack Overflow](http://stackoverflow.com/questions/12477823/calling-a-function-in-a-separate-file-in-python?answertab=votes#tab-top)  
(<http://stackoverflow.com/questions/12477823/calling-a-function-in-a-separate-file-in-python?answertab=votes#tab-top>)