

```
POptional.add_argument('--v', dest='verbose', action='count', default=0, help='increase verbosity level')
POptional.add_argument('--version', action='version', version='Version ' + str(constant.CURRENT_VERSION), help='laZagne version')

PWrite = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
PWrite._optionals.title = 'output'
PWrite.add_argument('-w', dest='write', action='store_true', help='write a text file on the current directory')

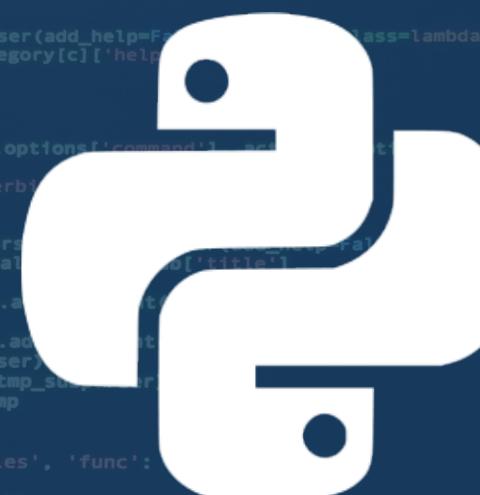
all_subparser = []
for c in category.keys():
    category[c]['parser'] = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
    category[c]['parser'].add_argument('--v', dest='verbose', action='count', default=0, help='increase verbosity level')
    category[c]['parser'].add_argument('--version', action='version', version='Version ' + str(constant.CURRENT_VERSION), help='laZagne version')

    category[c]['subparser'] = []
    for module in modules[c].keys():
        m = modules[c][module]
        category[c]['parser'].add_argument(m.options['command'], action='store_true', dest=m.options['dest'], help=m.options['help'])

        if m.suboptions and m.name != 'thunderbird':
            tmp = []
            for sub in m.suboptions:
                tmp_subparser = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
                tmp_subparser._optionals.title = sub['type']
                if 'type' in sub:
                    tmp_subparser.add_argument(sub['name'], action=sub['action'], dest=sub['dest'], help=sub['help'])
                else:
                    tmp_subparser.add_argument(sub['name'], action=sub['action'], dest=sub['dest'], help=sub['help'])
                tmp.append(tmp_subparser)
            all_subparser.append(tmp_subparser)
            category[c]['subparser'] += tmp

parents = [POptional] + all_subparser + [PWrite]
dic = {'all': {'parents': parents, 'help': 'Run all modules', 'func': None}}
for c in category.keys():
    parser_tab = [POptional, category[c]['parser']]
    if 'subparser' in category[c]:
        if category[c]['subparser']:
            parser_tab += category[c]['subparser']
    parser_tab += [PWrite]
    dic_tmp = {c: {'parents': parser_tab, 'help': 'Run %s module' % c, 'func': runModule}}
    dic = dict(dic.items() + dic_tmp.items())

subparsers = parser.add_subparsers(help='Choose a main command')
for d in dic.keys():
    subparsers.add_parser(d,parents=dic[d]['parents'],help=dic[d]['help']).set_defaults(func=dic[d]['func'],auditType=d)
```



# LISTAS

## PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

# CONCEPTO Y DEFINICIÓN

---

- Colección de datos ordenados
- Contienen cualquier tipo de dato: número, cadena, booleano, listas
- Sus valores van separados por coma, entre corchetes:

```
>>> e1 = [45, "lista", False, [100, 101]]
```

# OPERACIÓN: ACCEDER

---

- Para acceder a los elementos de una lista se escribe el nombre de la lista y se indica el número del índice entre corchetes:

```
>>> e2 = [True, False]
```

```
>>> val = e2[0]
```

```
>>> print(val)
```

- El índice del primer elemento de la lista es siempre 0, no 1

# OPERACIÓN: SLICING

---

- Es un mecanismo para seleccionar porciones de una lista
- En lugar de un índice, se indica el número de inicio y fin separados por dos puntos:

```
>>> e3 = [1, 2, 3, 4, 5]
```

```
>>> val1 = e3[0:2]
```

```
>>> print(val1)
```

- Se omite el número de la posición final

# OPERACIÓN: AGREGAR

---

- Se utiliza la función append ()

```
>>> e4 = [1, 2, 3]
```

```
>>> e4.append(4)
```

```
>>> print(e4)
```

```
[1, 2, 3, 4]
```

# OPERACIÓN: ELIMINAR

---

- Se utiliza la sentencia del

```
>>> e5 = ["matematica", "gramatica", "ciencias"]  
>>> print(e5)  
['matematica', 'gramatica', 'ciencias']  
  
>>> del e5[0]  
  
>>> print(e5)  
['gramatica', 'ciencias']
```

# OPERACIÓN: LONGITUD

---

- Para buscar el largo de la lista usamos la función len()

```
>>> e6 = [5, 10, 15, 20]
```

```
>>> print(len(e6))
```

4

# OPERACIÓN: CONCATENAR

---

- Para concatenar se utiliza el operador aritmético de suma

```
>>> e7 = [1, 2, 3]
```

```
>>> e8 = [4, 5, 6]
```

```
>>> print(e7 + e8)
```

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

```
POptional.add_argument('--v', dest='verbose', action='count', default=0, help='increase verbosity level')
POptional.add_argument('--version', action='version', version='Version ' + str(constant.CURRENT_VERSION), help='laZagne version')

PWrite = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
PWrite._optionals.title = 'output'
PWrite.add_argument('-w', dest='write', action='store_true', help='write a text file on the current directory')

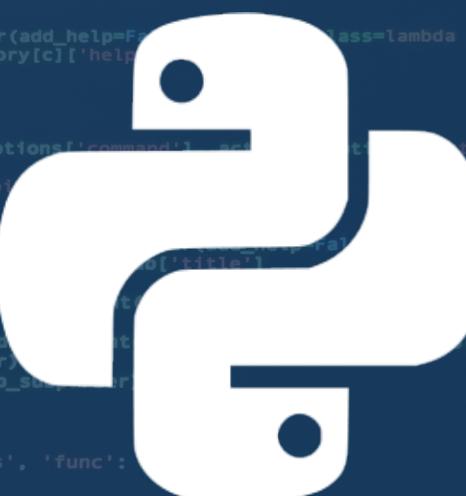
all_subparser = []
for c in category.keys():
    category[c]['parser'] = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
    category[c]['parser'].add_argument('--v', dest='verbose', action='count', default=0, help='increase verbosity level')
    category[c]['parser'].add_argument('--version', action='version', version='Version ' + str(constant.CURRENT_VERSION), help='laZagne version')

    category[c]['subparser'] = []
    for module in modules[c].keys():
        m = modules[c][module]
        category[c]['parser'].add_argument(m.options['command'], action='store_true', dest=m.options['dest'], help=m.options['help'])

        if m.suboptions and m.name != 'thunderbird':
            tmp = []
            for sub in m.suboptions:
                tmp_subparser = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
                tmp_subparser._optionals.title = sub['name']
                if 'type' in sub:
                    tmp_subparser.add_argument(sub['name'], type=sub['type'], action='store', dest=sub['dest'], help=sub['help'])
                else:
                    tmp_subparser.add_argument(sub['name'], action='store', dest=sub['dest'], help=sub['help'])
                tmp.append(tmp_subparser)
            all_subparser.append(tmp_subparser)
            category[c]['subparser'] += tmp

parents = [POptional] + all_subparser + [PWrite]
dic = {'all': {'parents': parents, 'help': 'Run all modules', 'func': runAll}}
for c in category.keys():
    parser_tab = [POptional, category[c]['parser']]
    if 'subparser' in category[c]:
        if category[c]['subparser']:
            parser_tab += category[c]['subparser']
    parser_tab += [PWrite]
    dic_tmp = {c: {'parents': parser_tab, 'help': 'Run %s module' % c, 'func': runModule}}
    dic = dict(dic.items() + dic_tmp.items())

subparsers = parser.add_subparsers(help='Choose a main command')
for d in dic.keys():
    subparsers.add_parser(d,parents=dic[d]['parents'],help=dic[d]['help']).set_defaults(func=dic[d]['func'],auditType=d)
```



# TUPLAS

## PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

# CONCEPTO Y DEFINICIÓN

---

- Secuencias de objetos, como las listas
- Son inmutables, es decir, no pueden ser cambiadas
- Sus valores van separados por coma, entre paréntesis:

```
>>> t1 = (65, "tupla", True)
```

# OPERACIÓN: ACCEDER

---

- Se hace igual que en las listas
- Para acceder a los elementos de una tupla se escribe el nombre de la tupla y se indica el número del índice entre corchetes:

```
>>> t2 = ("lunes", "miercoles", "viernes")
```

```
>>> print(t2[0])
```

lunes

```
>>> print(t2[1:3])
```

```
['miercoles', 'viernes']
```

# OPERACIÓN: ACTUALIZAR

---

- No es posible actualizar una tupla, son inmutables
- Se puede crear una nueva tupla basándose en una tupla existente

```
>>> t3 = (1, 2, 3)
```

```
>>> t4 = t3[0:1]
```

```
>>> print(t4)
```

```
(1,)
```

# OPERACIÓN: ELIMINAR

---

- ⦿ No se puede eliminar un elemento de una tupla
- ⦿ En su lugar, se borra toda la tupla

```
>>> t5 = ("calculo", "ortografia", "civica")
```

```
>>> del t5
```

```
>>> print(t5)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
NameError: name 't5' is not defined
```

# OPERACIÓN: LONGITUD

---

- Para buscar el largo de la tupla usamos la función len ()

```
>>> t6 = ('a', 'b', 'c')  
>>> print(len(t6))
```

3

# OPERACIÓN: CONCATENAR

---

- Para concatenar se utiliza el operador aritmético de suma

```
>>> t7 = (3, 6, 9)
```

```
>>> t8 = (2, 4, 6)
```

```
>>> print(t7 + t8)
```

```
(3, 6, 9, 2, 4, 6)
```

```
POptional.add_argument('--v', dest='verbose', action='count', default=0, help='increase verbosity level')
POptional.add_argument('--version', action='version', version='Version ' + str(constant.CURRENT_VERSION), help='laZagne version')

PWrite = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
PWrite._optionals.title = 'output'
PWrite.add_argument('-w', dest='write', action='store_true', help='write a text file on the current directory')

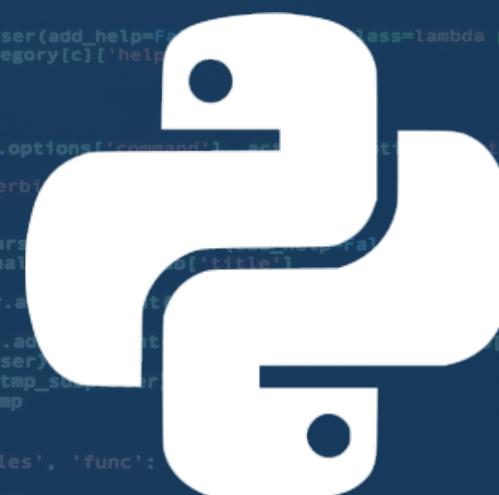
all_subparser = []
for c in category.keys():
    category[c]['parser'] = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
    category[c]['parser'].add_argument('--v', dest='verbose', action='count', default=0, help='increase verbosity level')
    category[c]['parser'].add_argument('--version', action='version', version='Version ' + str(constant.CURRENT_VERSION), help='laZagne version')

    category[c]['subparser'] = []
    for module in modules[c].keys():
        m = modules[c][module]
        category[c]['parser'].add_argument(m.options['command'], action='store_true', dest=m.options['dest'], help=m.options['help'])

        if m.suboptions and m.name != 'thunderbird':
            tmp = []
            for sub in m.suboptions:
                tmp_subparser = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
                tmp_subparser._optionals.title = sub['name']
                if 'type' in sub:
                    tmp_subparser.add_argument(sub['name'], type=sub['type'], action='store', dest=sub['dest'], help=sub['help'])
                else:
                    tmp_subparser.add_argument(sub['name'], action='store', dest=sub['dest'], help=sub['help'])
                tmp.append(tmp_subparser)
            all_subparser.append(tmp_subparser)
            category[c]['subparser'] += tmp

parents = [POptional] + all_subparser + [PWrite]
dic = {'all': {'parents': parents, 'help': 'Run all modules', 'func': None}}
for c in category.keys():
    parser_tab = [POptional, category[c]['parser']]
    if 'subparser' in category[c]:
        if category[c]['subparser']:
            parser_tab += category[c]['subparser']
    parser_tab += [PWrite]
    dic_tmp = {c: {'parents': parser_tab, 'help': 'Run %s module' % c, 'func': runModule}}
    dic = dict(dic.items() + dic_tmp.items())

subparsers = parser.add_subparsers(help='Choose a module')
for d in dic.keys():
    subparsers.add_parser(d,parents=dic[d]['parents'],help=dic[d]['help']).set_defaults(func=dic[d]['func'],auditType=d)
```



# DICCIONARIOS

## PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

# CONCEPTO Y DEFINICIÓN

---

- Colecciones que relacionan una clave y un valor  

```
>>> bib = {"Arte de la Guerra" : "Sun Tzu"}
```
- El primer valor es la clave y el segundo el valor asociado a la clave
- La clave puede ser cualquier tipo de dato y es inmutable

# OPERACIÓN: ACCEDER

---

- Se hace igual que en las listas y tuplas, con una diferencia
- Se acceden por su clave, no por su índice, entre corchetes:

```
>>> print(bib["Arte de la Guerra"] )
```

Sun Tzu

# OPERACIÓN: AGREGAR

---

- Debe escribirse el nombre del diccionario, seguido de la clave (entre corchetes), igualado al valor de la clave

```
>>> en_es[ 'puerta' ] = 'door'
```

# OPERACIÓN: LONGITUD

---

- Para ver la cantidad de valores se utiliza la función len ()

```
>>> es_en = {"casa" : "house", "perro" : "dog"}  
>>> len(es_en)
```

2

# OPERACIÓN: ELIMINAR

---

- Para eliminar un elemento del diccionario se usa la sentencia del
- El comando elimina la clave junto a su valor

```
>>> es_en = {"casa" : "house", "perro" : "dog"}
```

```
>>> del es_en["casa"]
```

- Para eliminar todo el diccionario usamos la función clear ()

```
>>> es_en.clear()
```

# OPERACIÓN: VALIDAR

---

- Se utiliza la sentencia `in`

```
>>> es_en = {"casa" : "house", "perro" : "dog"}  
>>> "casa" in es_en
```

True

```
POptional.add_argument('--v', dest='verbose', action='count', default=0, help='increase verbosity level')
POptional.add_argument('--version', action='version', version='Version ' + str(constant.CURRENT_VERSION), help='laZagne version')

PWrite = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
PWrite._optionals.title = 'output'
PWrite.add_argument('-w', dest='write', action='store_true', help='write a text file on the current directory')

all_subparser = []
for c in category.keys():
    category[c]['parser'] = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
    category[c]['parser'].add_argument('--v', dest='verbose', action='count', default=0, help='increase verbosity level')
    category[c]['parser'].add_argument('--version', action='version', version='Version ' + str(constant.CURRENT_VERSION), help='laZagne version')

    category[c]['subparser'] = []
    for module in modules[c].keys():
        m = modules[c][module]
        category[c]['parser'].add_argument(m.options['command'], action='store_true', dest=m.options['dest'], help=m.options['help'])

        if m.suboptions and m.name != 'thunderbird':
            tmp = []
            for sub in m.suboptions:
                tmp_subparser = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
                tmp_subparser._optionals.title = sub['name']
                if 'type' in sub:
                    tmp_subparser.add_argument(sub['name'], type=sub['type'], action='store', dest=sub['dest'], help=sub['help'])
                else:
                    tmp_subparser.add_argument(sub['name'], action='store', dest=sub['dest'], help=sub['help'])
                tmp.append(tmp_subparser)
            all_subparser.append(tmp_subparser)
            category[c]['subparser'] += tmp

parents = [POptional] + all_subparser + [PWrite]
dic = {'all': {'parents': parents, 'help': 'Run all modules', 'func': run_all}}
for c in category.keys():
    parser_tab = [POptional, category[c]['parser']]
    if 'subparser' in category[c]:
        if category[c]['subparser']:
            parser_tab += category[c]['subparser']
    parser_tab += [PWrite]
    dic_tmp = {c: {'parents': parser_tab, 'help': 'Run module %s' % c, 'func': runModule}}
    dic = dict(dic.items() + dic_tmp.items())

subparsers = parser.add_subparsers(help='Choose a main command')
for d in dic.keys():
    subparsers.add_parser(d,parents=dic[d]['parents'],help=dic[d]['help']).set_defaults(func=dic[d]['func'],auditType=d)
```

# FUNCIONES

## PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

# CONCEPTO

---

- Bloque de código organizado y reutilizable
- Sirve para realizar una tarea específica
- Proveen modularidad a la aplicación
- Python cuenta con sus funciones internas, como `print()`
- Sin embargo, el programador puede crear sus propias funciones

# REGLAS DE DEFINICIÓN

---

- Los bloques de función inician con la palabra reservada def seguido por el nombre de la función y paréntesis (())
- Cualquier parámetro de entrada, o argumento, debe ser definidos entre los paréntesis
- El bloque de código de la función inicia luego de los dos puntos (:) y debe ser debidamente sangrado
- Debe existir una expresión de retorno, return

# EJEMPLO

---

- El siguiente ejemplo toma un texto como argumento e imprime su valor en pantalla, sin retornar nada

```
def imprime(texto):  
    "Esta función imprime un texto"  
    print texto  
    return
```

# LLAMANDO UNA FUNCIÓN

---

- Definir una función solo da un nombre, argumentos y estructura del bloque de código a ejecutar
- El programador puede llamar una función desde otra función o bien globalmente
- Debe indicar el nombre de la función, seguido de paréntesis
- En caso de que tenga argumentos, también debe indicarlos

```
>>> imprime ("Curso")
```

# FUNCIONES ANÓNIMAS

---

- No son declaradas de la manera estándar
- Se utiliza la sentencia lambda
- Contienen una única expresión
- Acceden únicamente a sus variables definidas, no las globales
- No confundir con una función de una línea

# EJEMPLO

---

```
>>> sum = lambda arg1, arg2: arg1 + arg2;  
>>> print("Valor total: ", sum( 10, 20 ))  
Valor total: 30
```

# CONSIDERACIONES ESPECIALES

---

- Utilizar las funciones anónimas para tareas específicas que se cumplan en una sola sentencia y sean manejadores de llamados
- Utilizar las funciones tradicionales para tareas generales que serán reutilizables y modulares
- Si defino una variable dentro del bloque de código de la función sería una variable local a la función

```
POptional.add_argument('--v', dest='verbose', action='count', default=0, help='increase verbosity level')
POptional.add_argument('--version', action='version', version='Version ' + str(constant.CURRENT_VERSION), help='laZagne version')

PWrite = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
PWrite._optionals.title = 'output'
PWrite.add_argument('-w', dest='write', action='store_true', help='write a text file on the current directory')

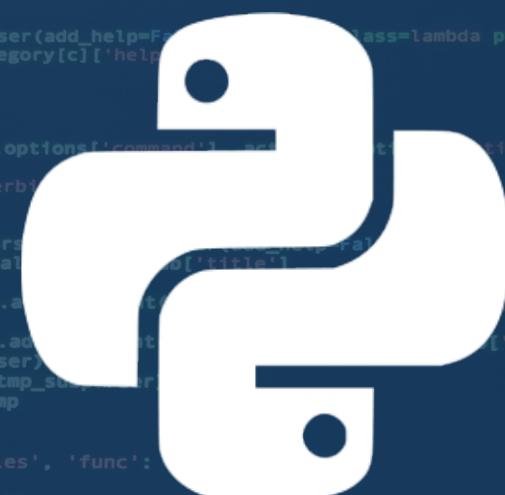
all_subparser = []
for c in category.keys():
    category[c]['parser'] = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
    category[c]['parser'].add_argument('--v', dest='verbose', action='count', default=0, help='increase verbosity level')
    category[c]['parser'].add_argument('--version', action='version', version='Version ' + str(constant.CURRENT_VERSION), help='laZagne version')

    category[c]['subparser'] = []
    for module in modules[c].keys():
        m = modules[c][module]
        category[c]['parser'].add_argument(m.options['command'], action='store_true', dest=m.options['dest'], help=m.options['help'])

        if m.suboptions and m.name != 'thunderbird':
            tmp = []
            for sub in m.suboptions:
                tmp_subparser = argparse.ArgumentParser(add_help=False,formatter_class=lambda prog: argparse.HelpFormatter(prog, max_help_position=constant.MAX_HELP_POSITION))
                tmp_subparser._optionals.title = sub['type']
                if 'type' in sub:
                    tmp_subparser.add_argument(sub['name'], type=sub['type'], action=sub['action'], dest=sub['dest'], help=sub['help'])
                else:
                    tmp_subparser.add_argument(sub['name'], action=sub['action'], dest=sub['dest'], help=sub['help'])
                tmp.append(tmp_subparser)
            all_subparser.append(tmp_subparser)
            category[c]['subparser'] += tmp

parents = [POptional] + all_subparser + [PWrite]
dic = {'all': {'parents': parents, 'help': 'Run all modules', 'func': None}}
for c in category.keys():
    parser_tab = [POptional, category[c]['parser']]
    if 'subparser' in category[c]:
        if category[c]['subparser']:
            parser_tab += category[c]['subparser']
    parser_tab += [PWrite]
    dic_tmp = {c: {'parents': parser_tab, 'help': 'Run module %s' % c, 'func': None}}
    dic = dict(dic.items() + dic_tmp.items())

subparsers = parser.add_subparsers(help='Choose a command to run')
for d in dic.keys():
    subparsers.add_parser(d,parents=dic[d]['parents'],help=dic[d]['help']).set_defaults(func=dic[d]['func'],auditType=d)
```



# EXCEPCIONES

## PROGRAMACIÓN DE COMPUTADORAS III

Abdel G. Martínez L.

# CONCEPTO Y DEFINICIÓN

---

- Es un evento que ocurre durante la ejecución de un programa que interrumpe el flujo normal de las instrucciones del programador
- Es un objeto Python que representa un error
- Cuando surge una excepción, se debe manejar inmediatamente o sino se terminará y cerrará el programa

# MANEJANDO UNA EXCEPCIÓN

---

- Si tenemos un código sospechoso que pueda levantar una excepción, entonces podemos defender nuestro programa con:
  - try
    - Abarca el bloque de código sospechoso
  - except
    - Bloque de código que maneja la excepción

# REGLAS

---

- Una sentencia de `try` puede tener múltiples sentencias `except`
- Se puede tener una cláusula genérica de `except`, para manejar cualquier tipo de excepción
- Luego de una cláusula `except`, se pueden incluir cláusulas `else` que se van a ejecutar si el bloque `try` no levanta una excepción
- El bloque `else` es un buen lugar para ubicar el código que no necesita ser protegido por el bloque `try`

# EJEMPLO

---

```
try:  
    fh = open("log.txt", "w")  
    fh.write("Manejo de Excepciones")  
  
except IOError:  
    print "Fallamos"  
  
else:  
    print "Tuvimos éxito!"  
    fh.close()
```

# LISTA DE EXCEPCIONES

---

Excepción	Descripción
Exception	Clase base de todas las excepciones
IndentationError	Inicia cuando el sangrado no está propiamente definido
ArithmetricError	Errores que ocurren en cálculos numéricos
OverflowError	Inicia cuando el cálculo excede el límite del tipo numérico
ZeroDivisionError	Inicia cuando se hace una división o módulo por cero
EOFError	Inicia cuando no hay entrada en input() y se alcanza un EOF
ImportError	Inicia cuando falla la sentencia de importación
IOError	Inicia cuando falla una operación de entrada/salida
RuntimeError	Inicia cuando el error no entra en ninguna categoría