



# Python

Antonio Insuasti

# **Parte 1**

Introducción al curso

Python big picture

Herramientas a utilizar

Primeros Pasos

Quien Soy??



## Antonio Insuasti

Wolfant

Ops engineer, developer, Crazy  
Joaco Dad, Gnu/Linux user @work  
Ops Architect & some days QA of  
production :)

@WolfantEc

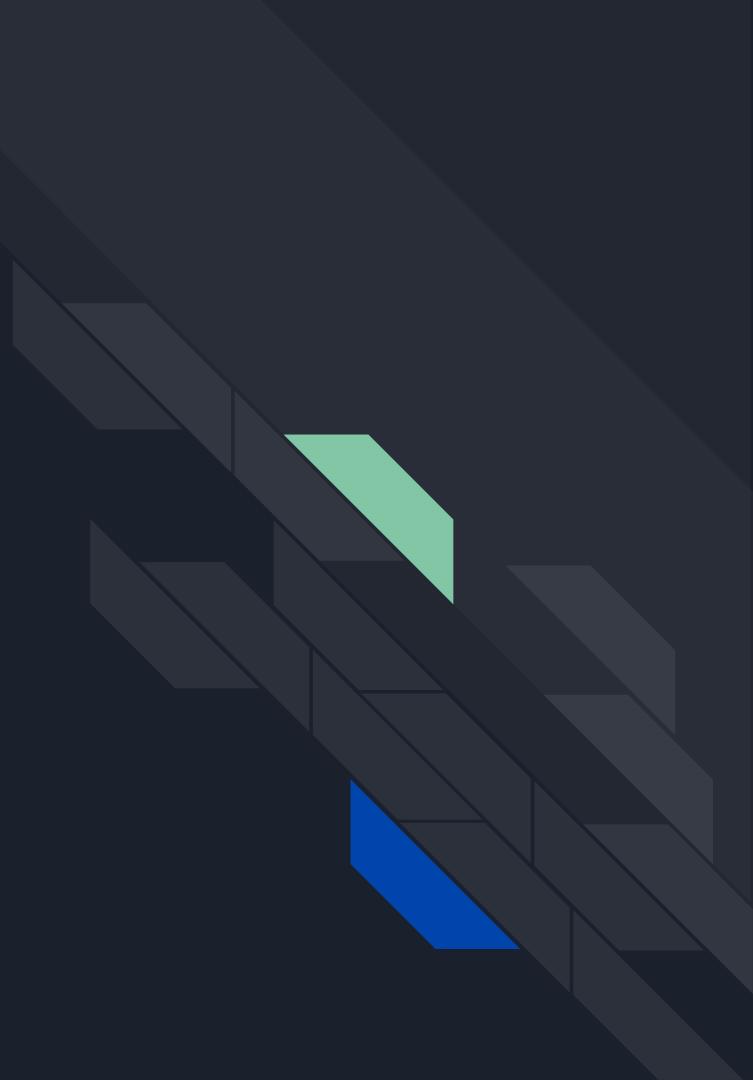
📍 Quito, Ecuador

🔗 <http://wolfant.insuasti.ec/>

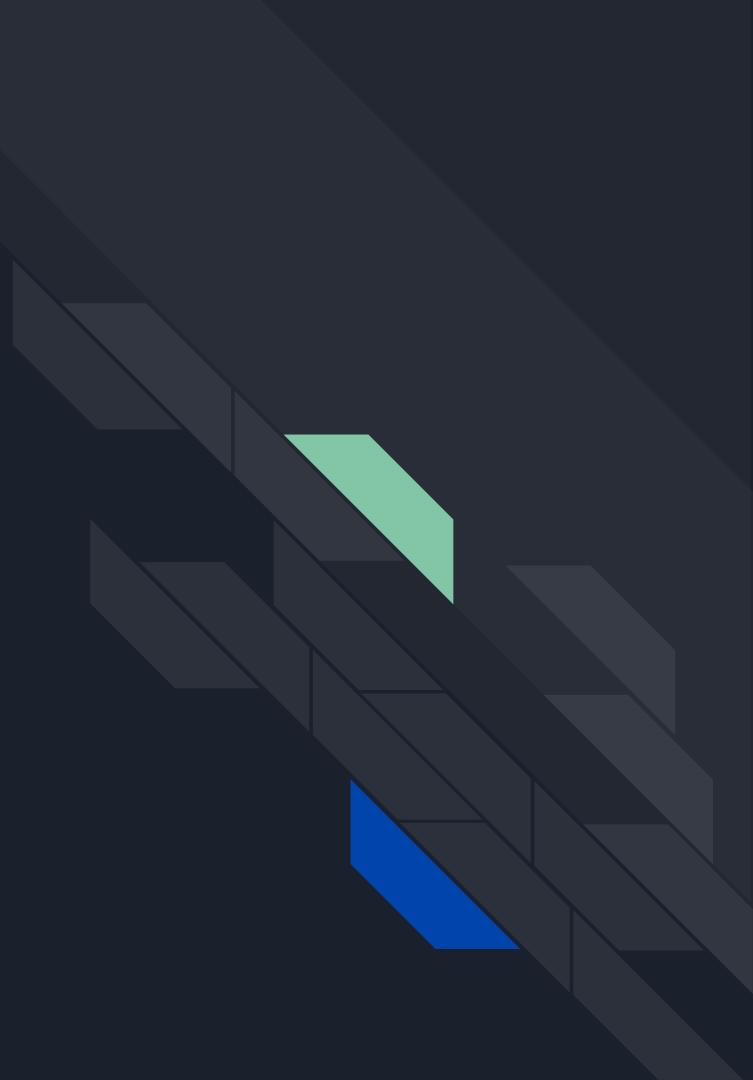
## Organizations



Quienes son ustedes?  
Por qué Python?



# Python Big Picture



# Qué es Python?

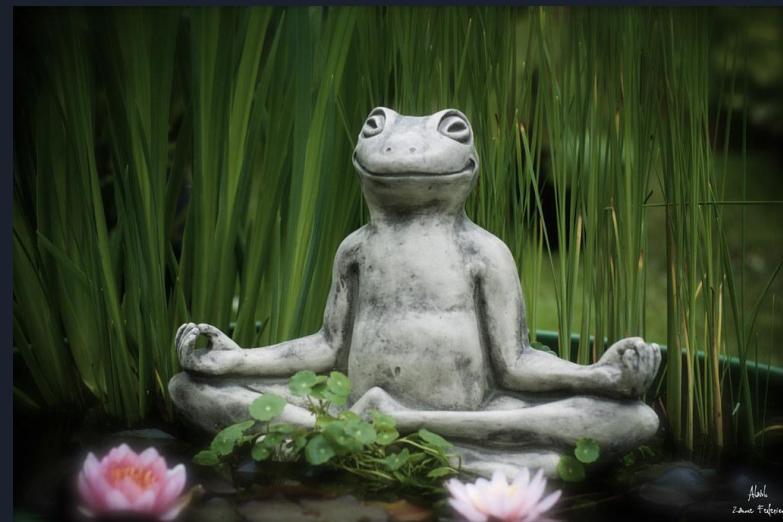
- Lenguaje de programación de alto nivel
- Dinámico (compilado e interpretado)
- Portable
- Poderoso
- Puede mezclarse (mixable)
- Fácil de aprender
- Fácil de entender/ leer
- Su nombre se debe a Monty Python
- Mantiene una Filosofía
- Orientado a Objetos y Funcional
- Libre
- PEP Python Enhancement Proposals



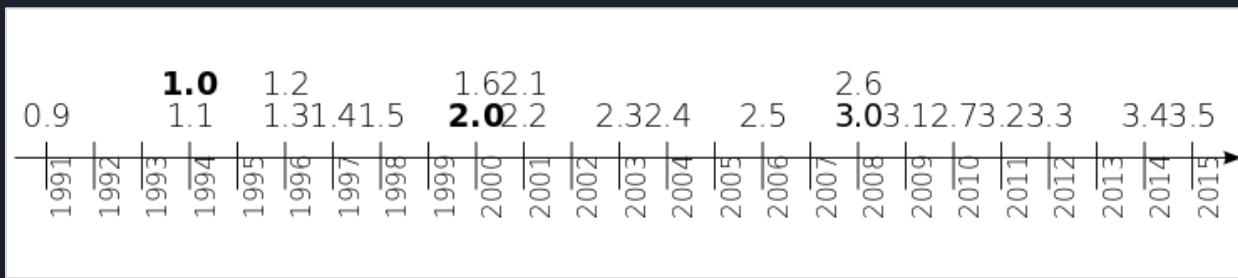
Guido Van Rossum  
Benevolent Dictator For Life" (BDFL)

# The Zen of Python (Pep20)

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!

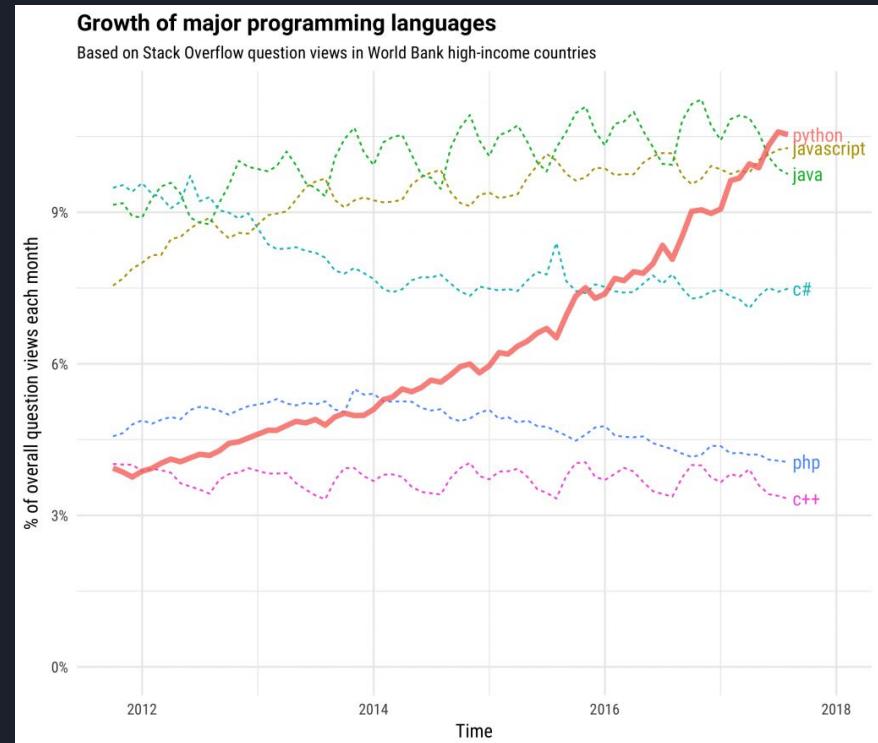


# Historia de Python

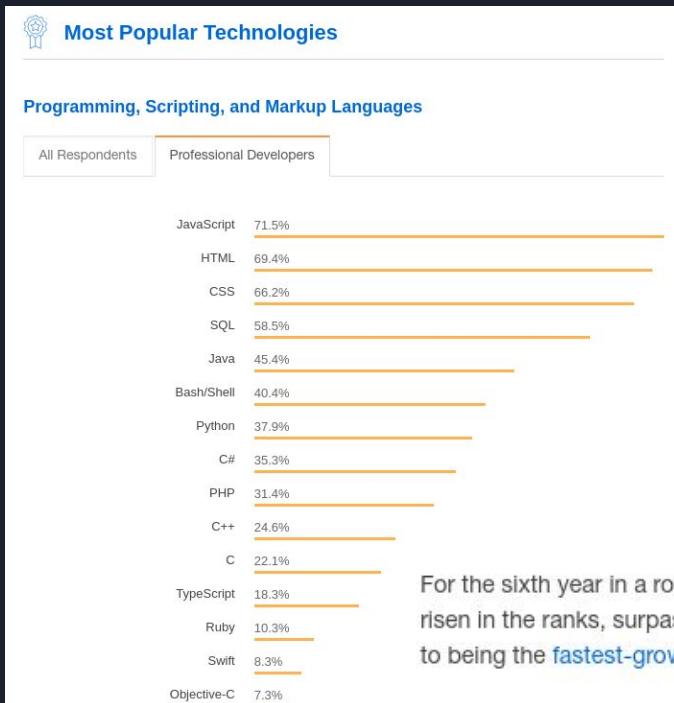


# Python hoy en día

- Web
- Scripting
  - Modelamiento 3D
  - Sistemas Operativos
- Programación científica y matemática
- Prototipado Rápido
- Juegos
- Machine Learning
- Robots (PyRo4)



# Python hoy en día



For the sixth year in a row, **JavaScript** is the most commonly used programming language. Python has risen in the ranks, surpassing C# this year, much like it surpassed PHP last year. Python has a solid claim to being the **fastest-growing major programming language**.



# Demo:

Fask - Django



# Como Funciona Python?

El intérprete:

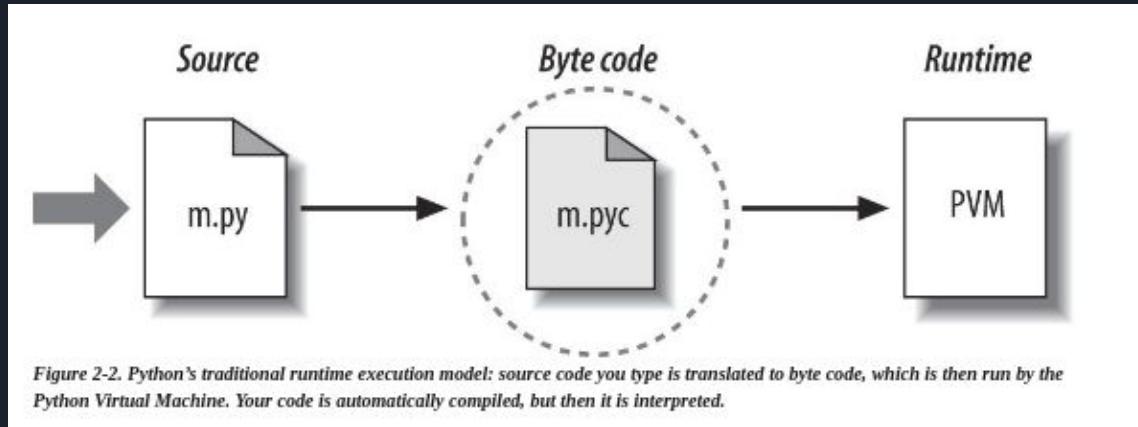
- Es un programa que ejecuta otros programas
- Una capa lógica entre el código y el hardware
- Convierte el código en byte code dependiendo de la implementación
  - CPython
  - Jython
  - IronPython
  - Stackless
  - PyPy

# Como Funciona Python?

PVM (Python Virtual Machine):

Una vez que el código ha sido compilado a byte code es enviado a ejecutarse en PVM

- El compilador siempre está presente en tiempo de ejecución y es parte del sistema que ejecuta los programas





# Demo:

Ejecución - Cache - Compilación



# Quiz

- Quien es BDFL?
- Qué es un interprete?
  - Por que PyPy es mucho más rápido que el compilador por defecto?
- Qué es el byte code?
- Qué es el código Fuente?
- Qué entiende por PVM
- Cual es la diferencia entre CPython, Jython y IronPython



# Primeros pasos

Conociendo Python



# Herramientas

- Distribución Linux (Fedora 28)
- Visual Studio Code
- Git
- Python 2 y 3
  - PIP
  - PEP8
  - REPL
  - bPython



# Jerarquización Conceptual

- Programas están compuestos por módulos
- los módulos contienen sentencias
- Sentencias contienen expresiones
- expresiones crean y procesan objetos



# Python REPL

## READ EVAL PRINT LOOP

- Debido a que el código es ejecutado inmediatamente el mejor lugar donde iniciar es la línea de comandos
- Interactive prompt
  - >>>
- Se recomienda el uso de bpython como intérprete



# Demo:

REPL

# Python Standard Library

Importar 3 formas de importar:

- `import {modulo}`
- `from {modulo} import { function }`
- `from { modulo } import { function } as {nombre}`

<https://docs.python.org/3.6/library/>



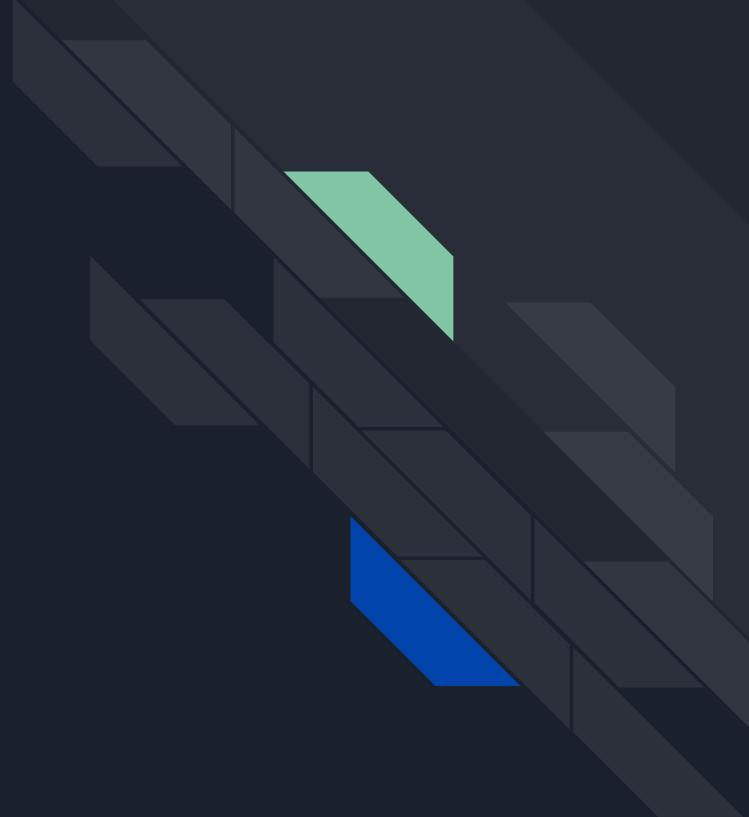


# Tipos Escalares y valores

# INT

12

Unlimited precision signed integer



# float

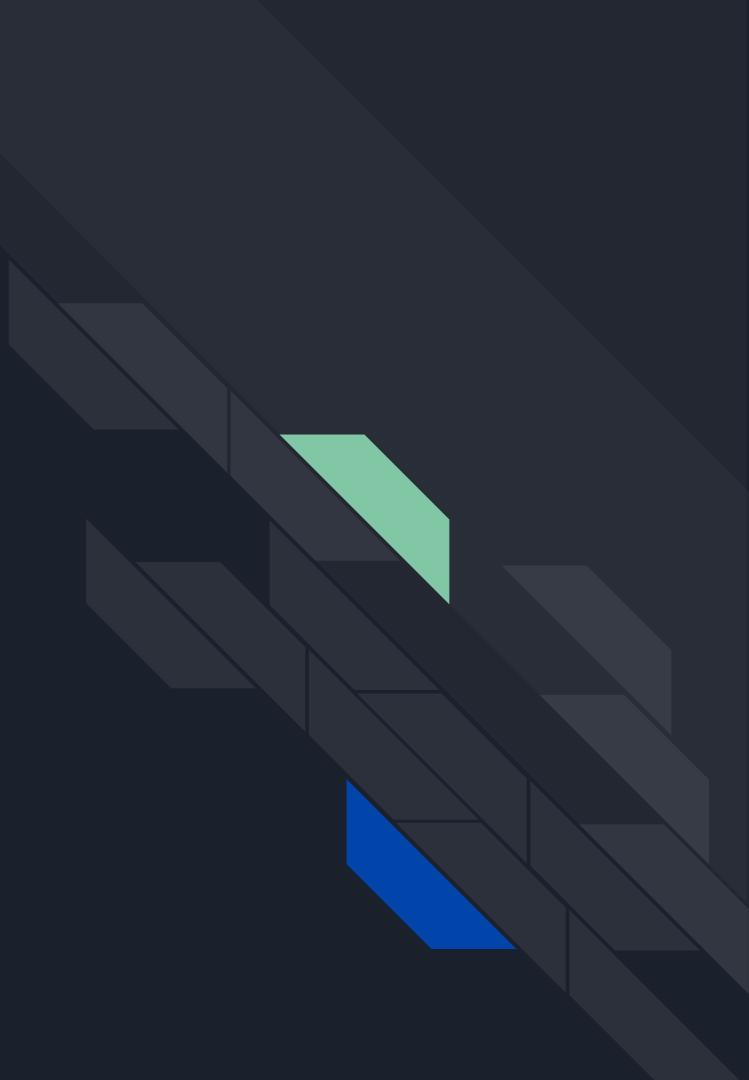
1.618

IEEE-754 double precision ( 64-bit)

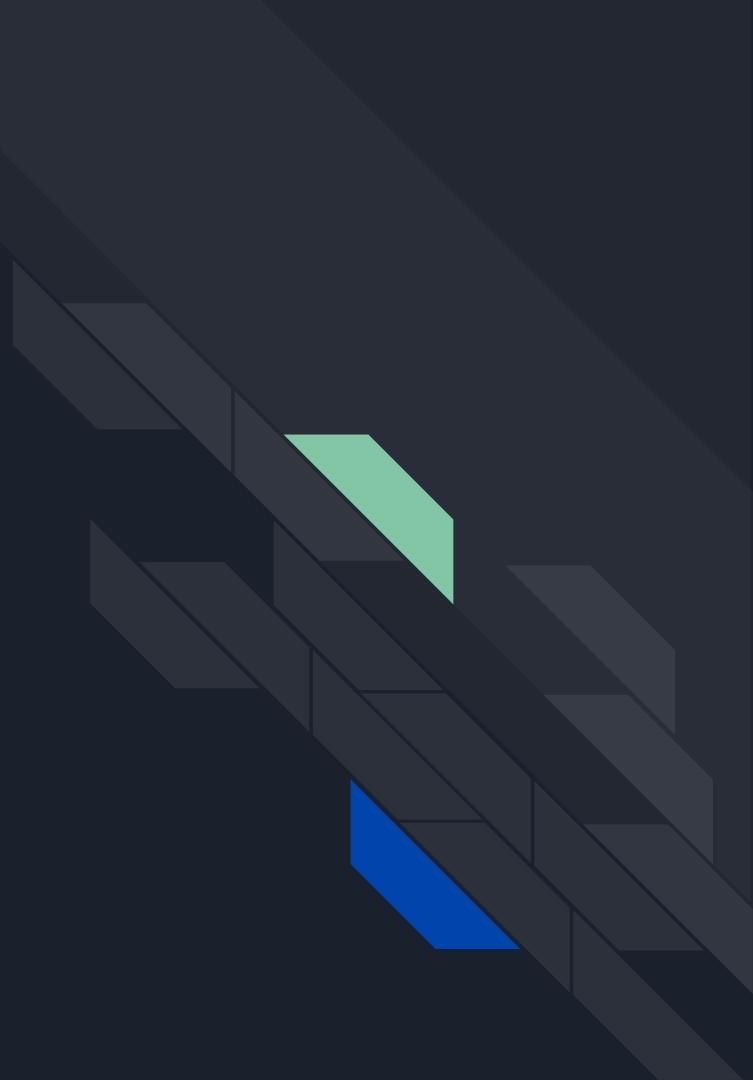
53 bits of binary precision

15 to 16 bits of decimal precision

nan inf numbers



# Nan & INF



# None

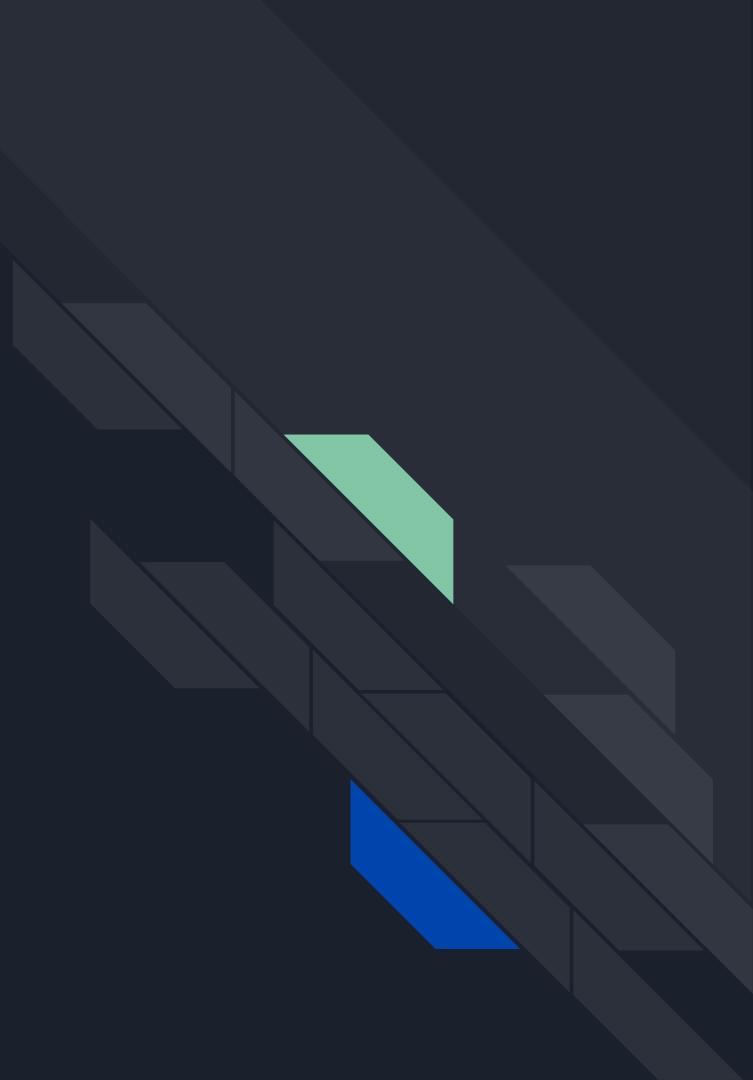
NoneType

Representa la ausencia de valor  
REPL no lo despliega

# Bool

Boolean Logical Value  
True | False

# Operadores



# Aritméticos

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

# Asignación

<b>Operator</b>	<b>Example</b>	<b>Same As</b>
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x  = 3	x = x   3



# Lógicos

Operator	Description	Example
and	Returns True if both statements are true	$x < 5$ and $x < 10$
or	Returns True if one of the statements is true	$x < 5$ or $x < 4$
not	Reverse the result, returns False if the result is true	<code>not(x &lt; 5 and x &lt; 10)</code>



# Identidad

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y



# Pertenencia

<b>Operator</b>	<b>Description</b>	<b>Example</b>
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

# Binarios

<b>Operator</b>	<b>Name</b>	<b>Description</b>
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off



# String y bytes

# Str

Immutable sequences of Unicode codepoints

“Hola” | ‘Hola’ |





# Strings

- Multi línea
  - tres comillas ""
  - Universal New Line PEP 278
- Con secuencias de escape
  - RAW Strings
- Soporta funciones como un objeto de una clase
  - Ejm. string.capitalize()

Mas info en:

[https://docs.python.org/3/reference/lexical\\_analysis.html#string-and-bytes-literals](https://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals)

# bytes

Immutable sequences of bytes

```
b“Hola” | b ‘Hola’ |
```





# bytes

Funcionan como un string

- Archivos
- Recursos de Red
- Respuestas HTTP



# Listas

Es una secuencia de objetos mutables

- Permite asignaciones y eliminaciones
- Se usan los brackets para denotar listas, se separan por comas
- Tiene un constructor

```
[1, "a", 3] | [1, "a", 3]
```



# Tuples

Es una secuencia de objetos inmutables

- Se usan los paréntesis para denotar, se separan por comas
- El uso que se les da es la inmutabilidad
- Tiene un constructor

(1, "a", 3) | (1, "a", 3,) | (1,)



# Diccionarios

mapeo mutable de llaves y valores

- Permite asignaciones y eliminaciones
- Se usan los llaves para denotar diccionarios, se separan por comas, y se asignan con dos puntos
- Tiene un constructor

```
{'Antonio': 34, 'Juan': 35, 'Bob': 31}
```



# Sentencias Primitivas

if while for



# IF | ELSE | ELIF

Condicionales:

- Igual:  $a == b$
- No es igual :  $a != b$
- Menor:  $a < b$
- Menor o igual:  $a <= b$
- Mayor:  $a > b$
- Mayor o igual:  $a >= b$



# IF | ELSE | ELIF

```
if expression:  
    {TO-DO}
```

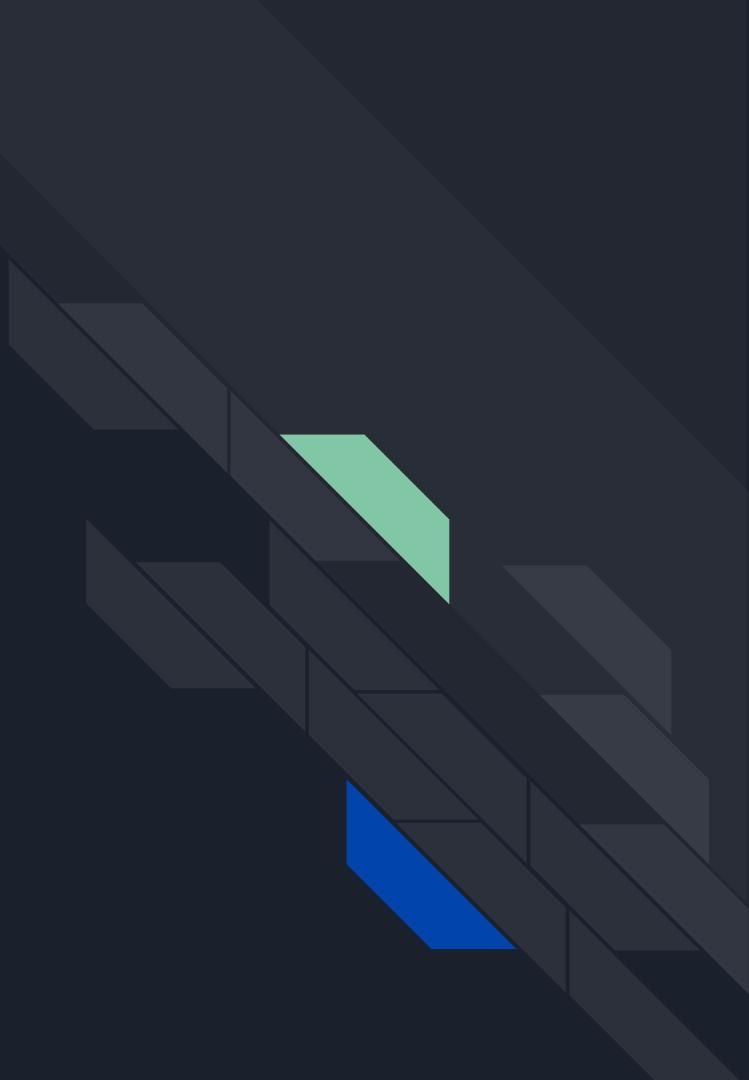
```
if expression:  
    {TO-DO}  
else  
    {TO-DO}
```

```
if expression:  
    {TO-DO}  
else:  
    if expression:  
        {TODO}
```

```
If expression:  
    {TO-DO}  
elseif expression:  
    {TO-DO}  
else  
    {TO-DO}
```

```
if expression: {TO-DO}  
{TO-DO} if expression else {TO-DO}  
{TO-DO} if expression else {TO-DO} if expression else {TO-DO}
```

Flat is better than  
nested





# While

```
while expression:  
    {TO-DO}
```

```
c = 5  
while c != 0:  
    print(c)  
    c -= 1
```

```
while expression:  
    {TO-DO}  
    break | continue
```

```
c = 5  
while c:  
    print(c)  
    c -= 1
```



# While

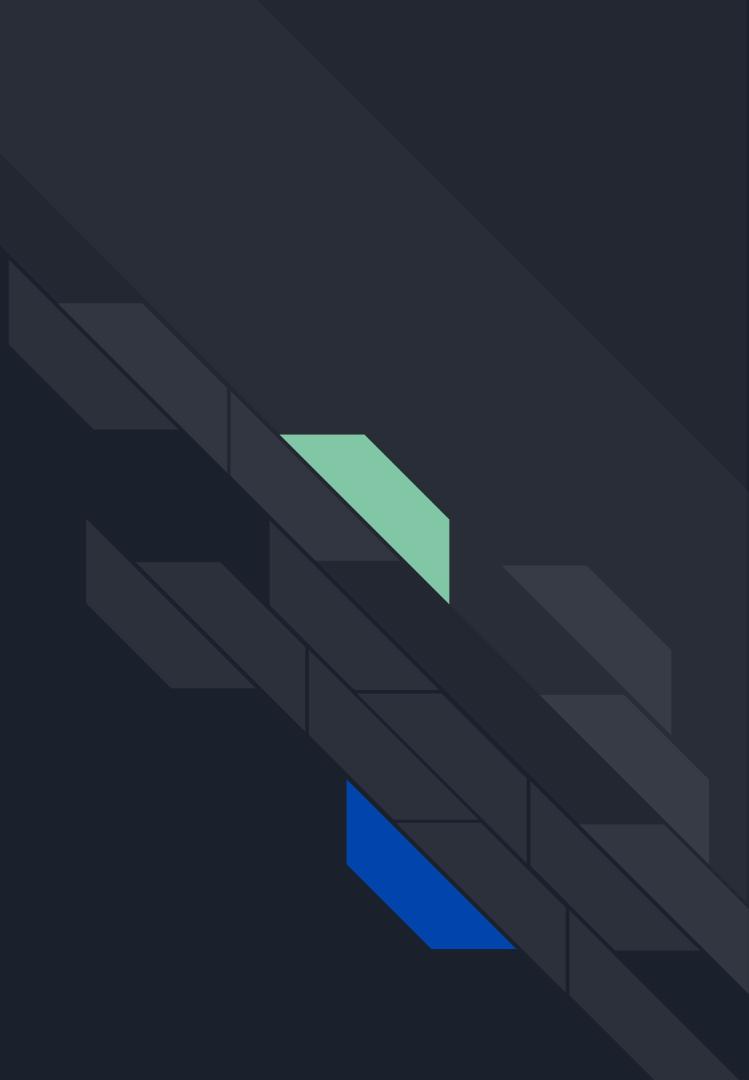
while expression:  
{TO-DO}

```
c = 5  
while c != 0:  
    print(c)  
    c -= 1
```

while expression:  
{TO-DO}  
break | continue

```
c = 5  
while c:  
    print(c)  
    c -= 1
```

Explicit is better than  
implicit.





# For

```
for {var|var,...} in interator():
    {TO-DO}
```

función range:  
range({start}, {stop}, {step})

```
for {var|var,...} in interator():
    {TO-DO}|break
```

```
for {var|var,...} in interator():
    {TO-DO}
else:
    {TO-DO}
```

```
for {var|var,...} in interator():
    for {var|var,...} in interator():
        {TO-DO}
```



# Comprehensiones Nivel Básico

Procesan estructuras de datos

[expr(item) **for** item **in** iterable]

M = [[1,2,3], [4,5,6], [7,8,9,]]

- [row[1] for row in M]
- [q[1] \* 10 for q in M]
- [q[1] for q in M if q[1] % 2 == 0]



# Práctica - Todo Junto

primitivas



# Práctica

Casos de Ejemplo



# Prácticas

Ingreso un numero con input y ver si es par o impar

Ingreso de un vector de 5 números ordenar de menor a mayor y de mayor a menor

Imprima todos los número impares del 1 al 100 y muestre la suma

función que sume dos objetos

dado el diccionario: dict = { 'Antonio': 34 , 'Juan': 35 , 'Bob': 31 }, sume la edad



# Modularidad

# Módulos

```
def _int32_to_bytes(i):
    """Convert an integer to four bytes in little-endian format."""
    return bytes( (i & 0xff,
                  i >> 8 & 0xff,
                  i >> 16 & 0xff,
                  i >> 24 & 0xff) )
```

```
def _bytes_to_int32(b):
    """Convert a bytes object containing four bytes into an integer."""
    return b[0] | (b[1] << 8) | (b[2] << 16) | (b[3] << 24)
```

```
def fetch_words():
    with urlopen('http://sixty-north.com/c/t.txt') as story:
        story_words = []
        for line in story:
            line_words = line.decode('utf8').split()
            for word in line_words:
                story_words.append(word)
    return story_words
```

```
def print_items(items):
    for item in items:
        print(item)
```

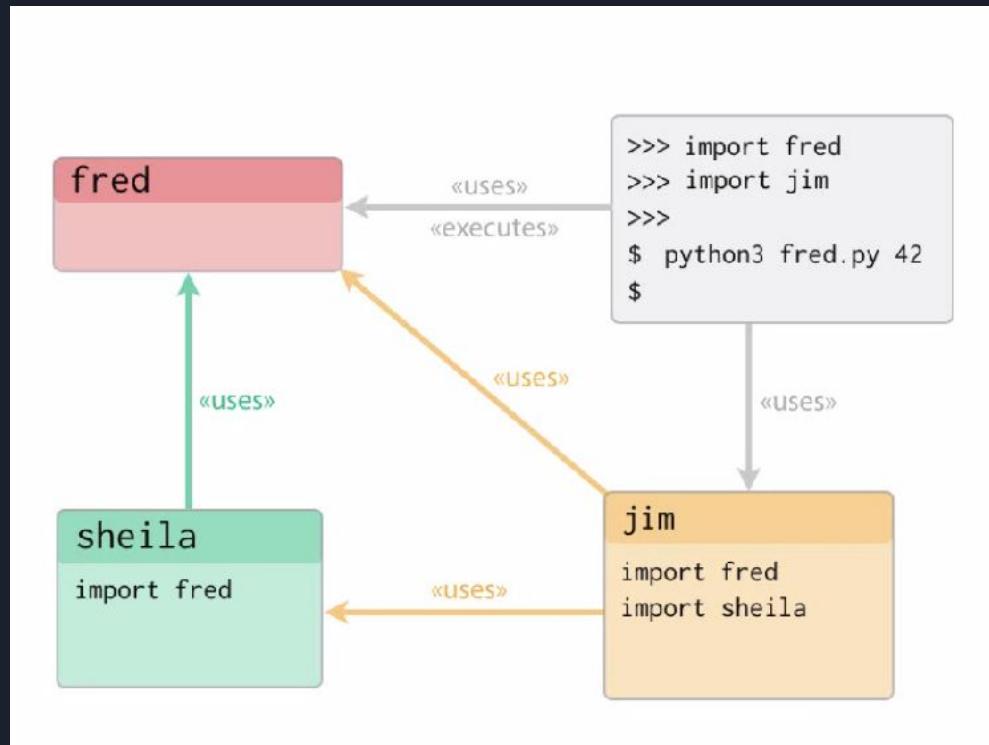
```
def main():
    url = sys.argv[1]
    words = fetch_words(url)
    print_items(words)
```

```
def console_card_printer(passenger, seat, flight_number, aircraft):
    output = "| Name: (0)" \
            " Flight: (1)" \
            " Seat: (2)" \
            " Aircraft: (3)" \
            "|".format(passenger, flight_number, seat, aircraft)
banner = '+' + '-' * (len(output) - 2) + '+'
border = '| ' + ' ' * (len(output) - 2) + '|'
lines = [banner, border, output, border, banner]
card = '\n'.join(lines)
print(card)
print()
```

```
def make_flight():
    f = Flight("BA758", Aircraft("G-EUPT", "Airbus A319",
                                  num_rows=22, num_seats_per_row=6))
    f.allocate_seat('12A', 'Guido van Rossum')
    f.allocate_seat('15F', 'Bjarne Stroustrup')
    f.allocate_seat('15E', 'Anders Hejlsberg')
    f.allocate_seat('1C', 'John McCarthy')
    f.allocate_seat('1D', 'Richard Hickey')
    return f
```



# Módulos





# Funciones

- Se definen con la palabra `def` y terminan en :
- Deben tener un nombre, el cual debe ser escrito en minúsculas, palabras separadas por `_`
  - Se puede usar camelCase en caso de compatibilidad (pep8)
- Después de definir una función se debe indentar

```
def square(x)
    return x * x
```

- `return` sin parámetros o un `return` implícito retorna `None`



# Funciones

Se debe distinguir entre usar y ejecutar

- from modulos3 import fetch\_words as fs
- python3 modulos3.py
- Atributos y variables especiales en Python son denotados con dos underscore \_\_
  - \_\_name\_\_
  - \_\_main\_\_
  - \_\_init\_\_



# Modelo de Ejecución

Cuando las funciones son definidas?

Que pasa cuando un módulo es importado?

# Modelo de Ejecución

Python module

Convenient import with API

Python script

Convenient execution from  
command line

Python program

Perhaps composed of many  
modules

Python module  
Python script

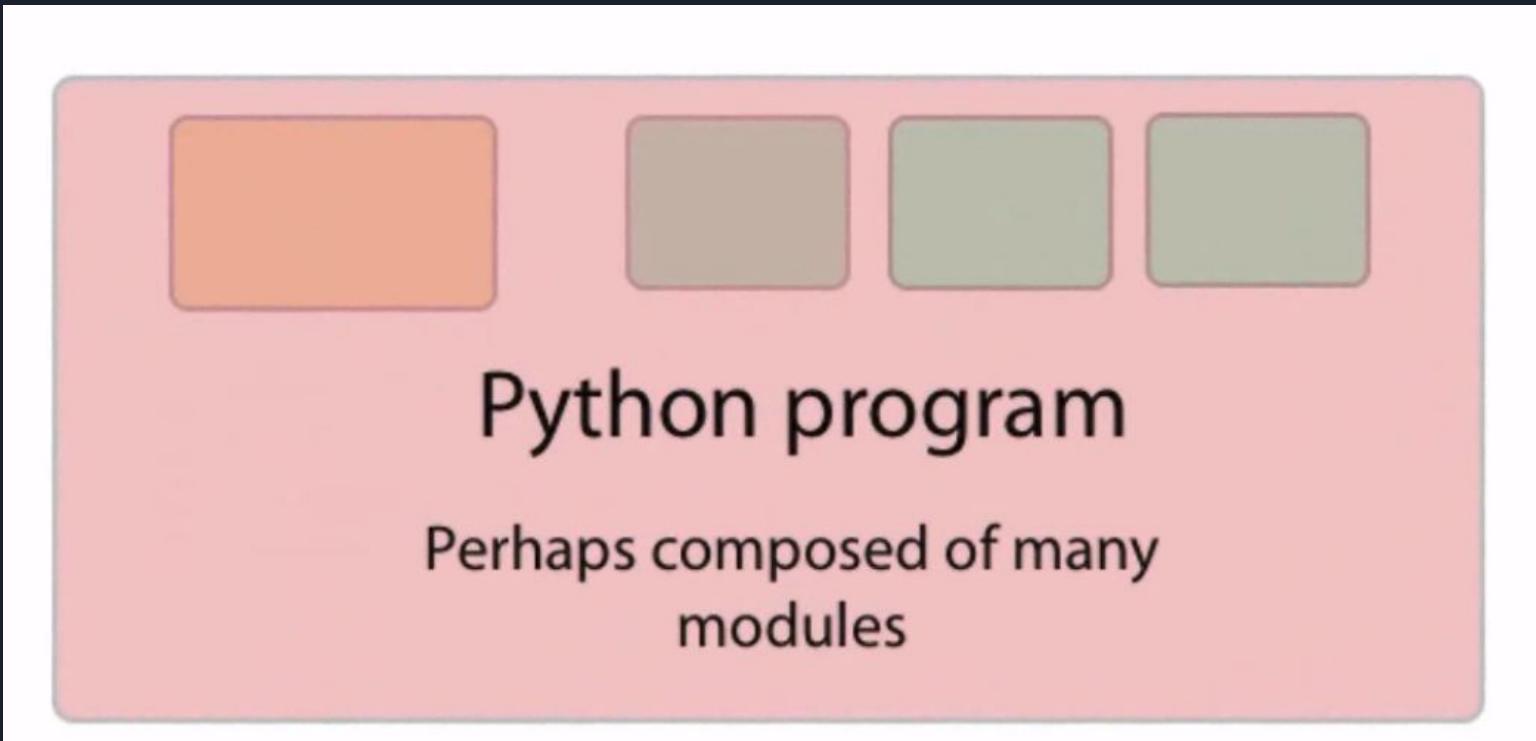
Convenient execution from  
command line  
Convenient import with API

Python program

Perhaps composed of many  
modules



# Modelo de Ejecución



# Sparse is better than dense.

- pep8 recomienda 2 líneas entre funciones



# Documentación

docstring



# Docstring

Convenciones:

- PEP 257 ( no es ampliamente usado)
- reStructuredText -> Sphinx
- Google Python Style Guide

Se utiliza el # para comentar líneas

Se debe tener el python en el path y usar el shebang `#!/$PATH_PYTHON`

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

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

```
which python3
```



# Objetos y Argumentos



# Objetos y Argumentos

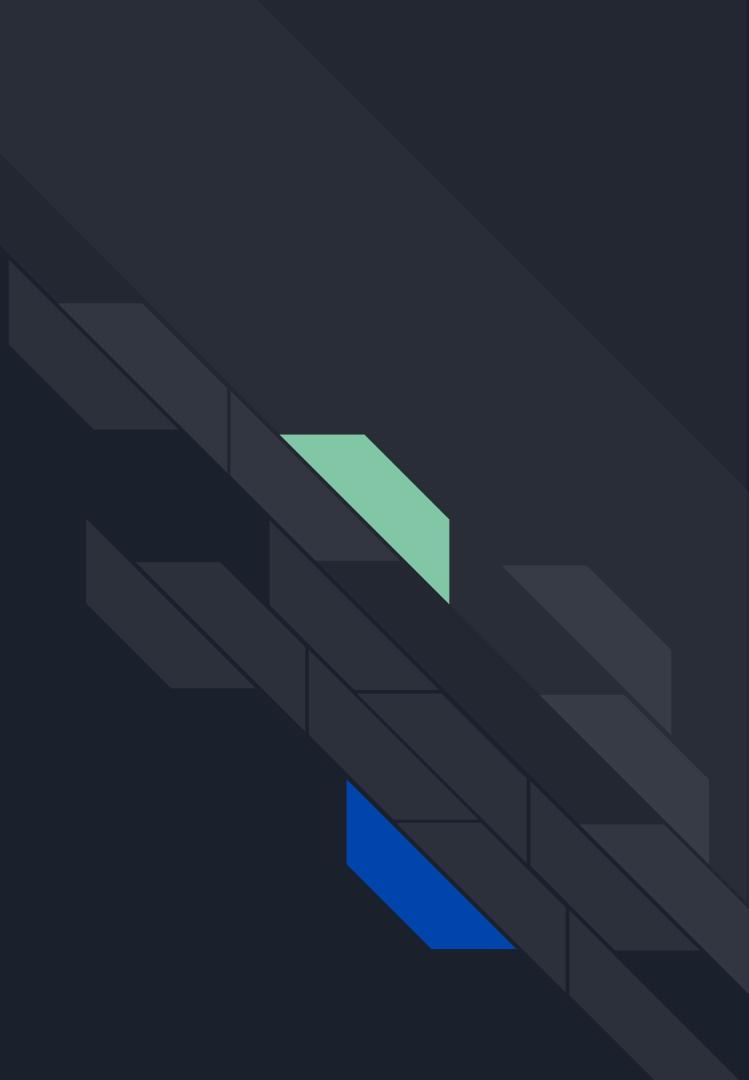
- un objeto puede ser idéntico a otro
- un objeto puede tener el mismo valor que otro sin necesidad de ser idéntico.
  - función id()
- un objeto sin referencia es eliminado por el GC
- un objeto entra a una función por referencia, no por valor
- es responsabilidad de la función hacer una copia del objeto para ser modificado internamente
- Los argumentos son posicionales excepto si se define el nombre
- los argumentos por defectos son evaluados cuando def es evaluado una sola vez y puede ser modificados como cualquier objeto
- los argumentos por defectos son inicializados y referenciados cuando def es evaluado.
- Python es dinámico y tipado



# Python name scopes

- Local: dentro de una función
  - Enclosing: todas las funciones
  - Global: top-level en módulos
  - Built-in: Provisto por módulos builtins
- 
- Module Cope Name
    - sys
    - fetch\_words
    - print\_items
    - main
  - Local Scope Names
    - story
    - story\_words
    - line
    - line\_words
    - words

Special cases aren't  
special enough to break  
the rules.





# Packing & Unpacking

De variables:

- se usa la coma ,
- debe ser el mismo número de un return o datos de una tupla diccionario

En argumentos:

- \* para tuplas
- \*\* para diccionarios



# STR functions

- Join
- split
- format
  - unpack
- partition

the way may not be obvious at first

- para concatenar invoca join en un texto vacío. crear algo de la nada
- es natural y elegante



# Práctica

Módulo para validación de contraseñas que cumpla con los siguientes criterios de aceptación:

- debe contener un mínimo de 8 caracteres.
- debe contener letras minúsculas, mayúsculas, números y al menos 1 carácter no alfanumérico.
- no puede contener espacios en blanco.
- si es válida, devuelve "contraseña validada correctamente" ..
- si no válida, devuelve "La contraseña elegida no es segura".



# Prácticas

Escribir una función que tome un carácter y devuelva True si es una vocal, de lo contrario devuelve False.

Módulo para validación de nombres de usuarios que cumpla con los siguientes criterios de aceptación:

- debe contener un mínimo de 8 caracteres y un máximo de 15.
- debe ser alfanumérico.
- con menos de 8 caracteres, devuelve el mensaje "El nombre de usuario debe contener al menos 8 caracteres".
- con más de 12 caracteres, devuelve el mensaje "El nombre de usuario no puede contener más de 15 caracteres".
- con caracteres distintos a los alfanuméricos, devuelve el mensaje "El nombre de usuario puede contener solo letras y números".
- si es válido, devuelve "nombre de usuario validado correctamente".



# Prácticas

Crear una función que reciba una frase por pantalla e indicar cuántas vocales y cuantas palabras tiene la frase, debe retornar los dos valores

Función que convierta números binarios en enteros.



# Prácticas

Ingresar una variable numérica, imprimir un mensaje en pantalla que indique si esta variable numérica está entre 0 y 10.

Mediante el uso de "while" mostrar los números del 1 al 100.

Mediante el uso de "for" mostrar los números del 1 al 100.

Ingresar dos cadenas por teclado, mostrar ambas cadenas con un espacio entre ellas y con los 2 primeros caracteres intercambiados. Por ejemplo, "hola mundo" pasaría a "mula hondo"



# Tarjeta de Embarque

Desarrollar una función que reciba los datos de:

Número de Vuelo, Aerolínea, Origen, Destino, Número de Asiento, Nombre, Apellido, Edad del Ocupante y que lo imprima con un banner hecho en consola con el nombre de la empresa vendedora de boleto

# Range

- Secuencia aritmética - progresión de int
- Inicia desde 0
- permite crear una lista list(range(5,10))
- en REPL no imprime ningún valor

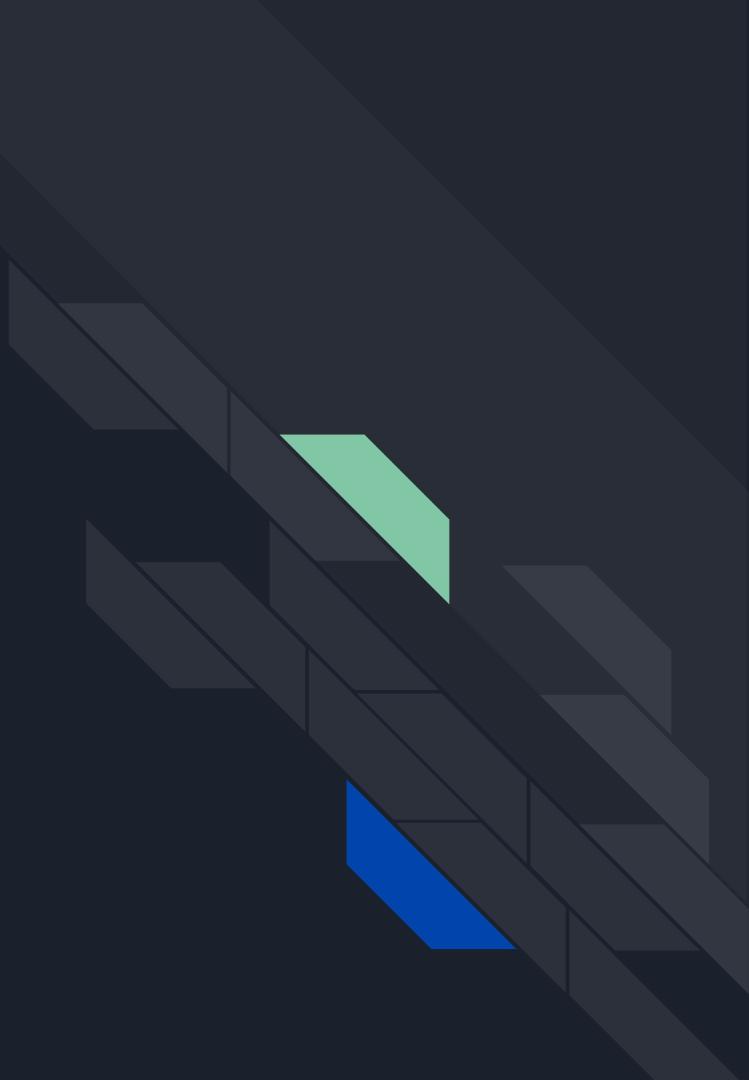
```
range(0,5)
for i in range(5):
    print(i)
```

range(Start,Stop,salt)

- start incluye
- stop no incluye

Constructor	Arguments	Result
range(5)	stop	0, 1, 2, 3, 4
range(5, 10)	start, stop	5, 6, 7, 8, 9
range(10, 20, 2)	start, stop, step	10, 12, 14, 16, 18

# Listas Diccionarios Sets avanzado



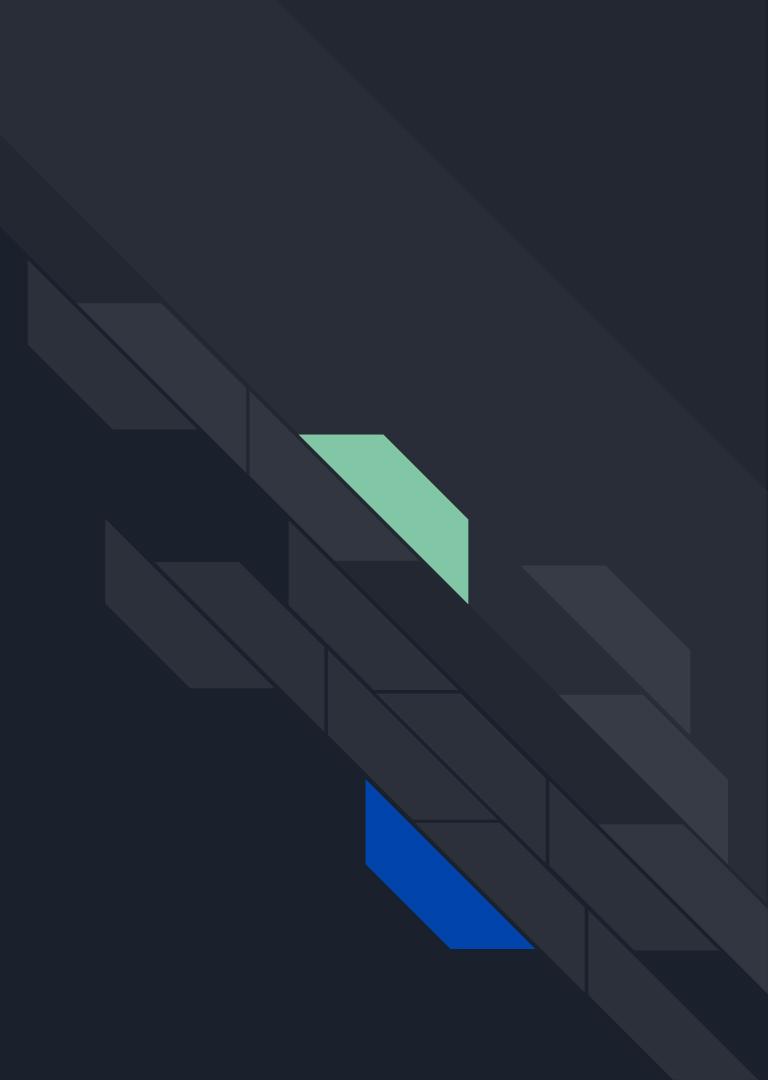


# Recortar Listas

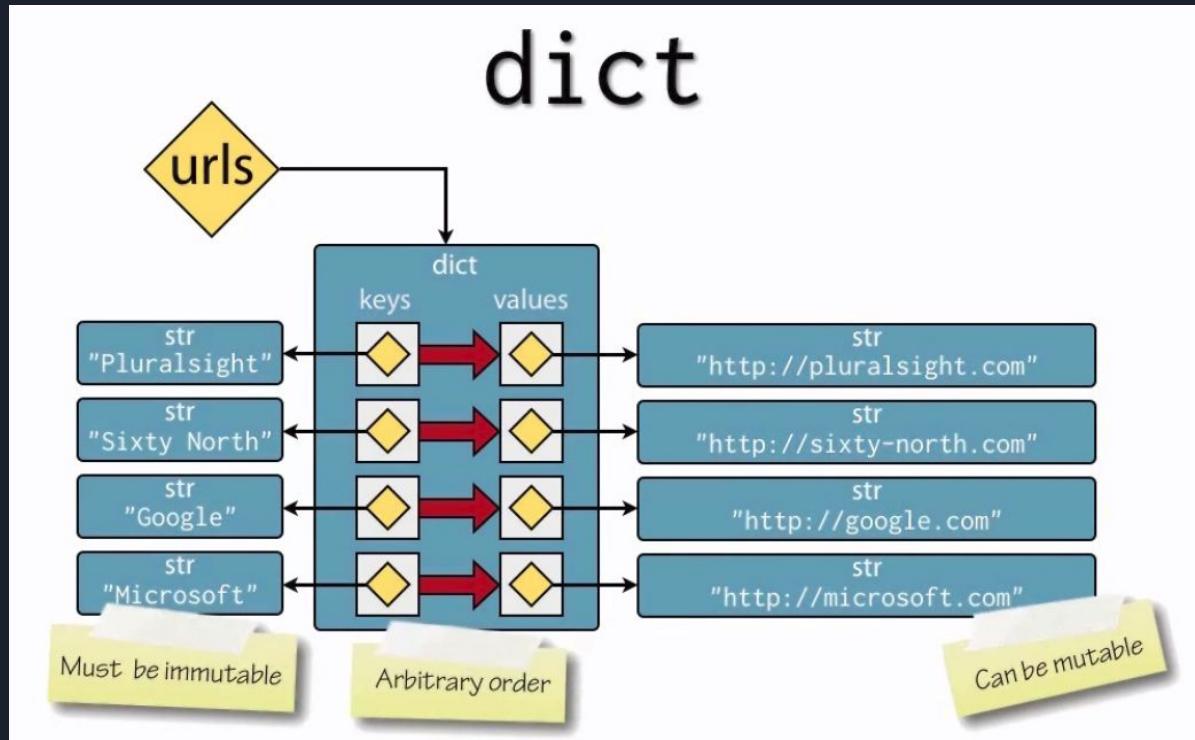
```
s = ['show', 'how', 'to', 'index', 'into', 'sequences']
```

- s[start:stop]
  - stop no incluye
- Contiene index positivos y negativos lo que le hace elegante
  - s[5] == s [-1]
  - evitar: seq[len(seq) - 1 ]

# Copias superficiales (shallow copies) y funciones de lista

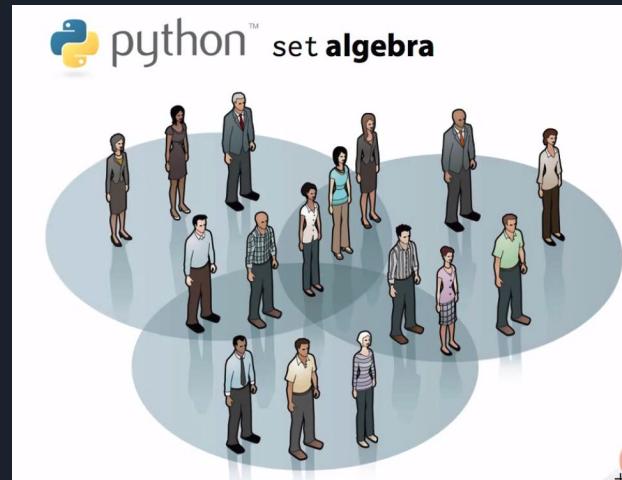


# Diccionarios



# SET

- Colección de objetos únicos e inmutables
- se usa {} para identificar set, objeto separado por
- es iterable
- mantiene membresía (in , no in )
- para agregar un miembro se usa object.add(new\_object)
- para eliminar se usa .remove o .discard
- Principal uso para matemáticas y álgebra
  - s.union(t)
  - s.intersection(t)
  - s.difference(t)
  - s.symmetric\_difference(t)
  - s.issubset(t)



# Protocolo de Colecciones - PEP 544

Protocol	Implementing Collections
<b>Container</b>	str, list, range, tuple, set, bytes, dict
<b>Sized</b>	str, list, range, tuple, set, bytes, dict
<b>Iterable</b>	str, list, range, tuple, set, bytes, dict
<b>Sequence</b>	str, list, range, tuple, set, bytes
<b>Mutable Sequence</b>	list
<b>Mutable Set</b>	set
<b>Mutable Mapping</b>	dict



# Prácticas

Crear un módulo con varias funciones para manejo de listas y diccionarios que realicen las siguientes acciones:

Agregar un elemento al final de la lista/Diccionario, devolver "el elemento fue agregado".

Eliminar el último elemento, devolver "el elemento fue eliminado".

Ordenar en forma Descendente, debe servir para números o letras

Agregar o eliminar un elemento en la ubicación que el usuario requiera. Al agregar un elemento en una ubicación ocupada, el diccionario/lista deberá expandirse de forma dinámica



# Manejo de excepciones

Excepciones y control de flujo



# Manejo de excepciones?

Manejo de excepciones es un mecanismo para parar el flujo “normal” de un programa y continuar en un contexto circundante o un bloque de código diferente

(Raise) enviar|botar|crear una excepción que interrumpe el flujo del programa

(handle) manejar una excepción para retomar el control del programa

(unhandled exceptions) terminan la ejecución del programa

(Exception Objects) contienen información sobre el evento de excepción

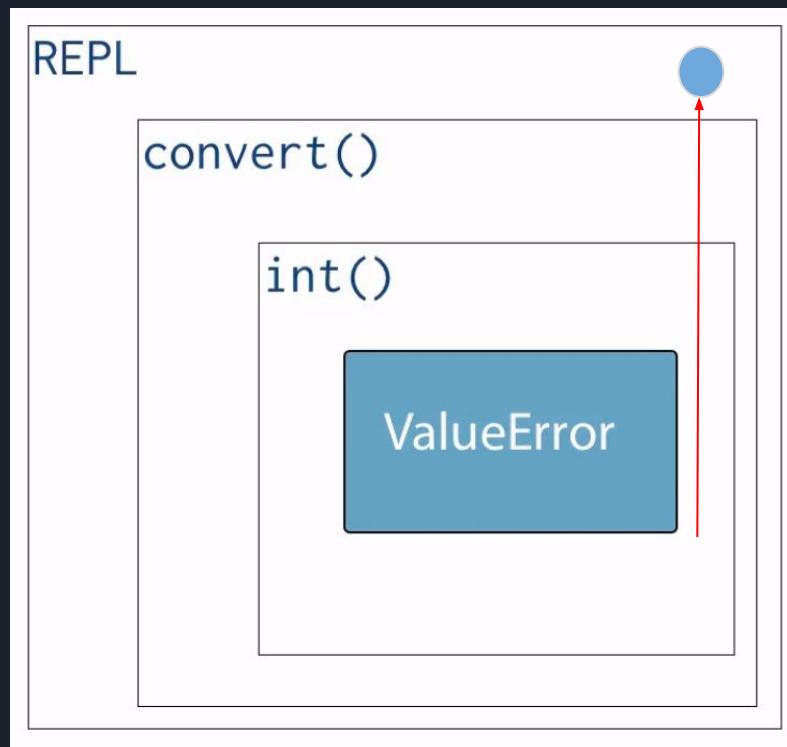


# Manejo de excepciones

```
>>> s = cv('hola mundo')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/antonio/GIT/cursoPython/Parte1/Exceptions/exceptions1.py", line 3, in convert
    x = int(s)
ValueError: invalid literal for int() with base 10: 'hola mundo'
```

ValueError: Error message

# Manejo de excepciones



# Demo



# Manejo de excepciones

```
def convert(s):
    '''Convert to an integer.'''
    try:
        x = int(s)
    except ValueError:
        x = -1
    return x
```

# Manejo de excepciones

```
def convert(s):
    '''Convert to an integer.'''
    try:
        x = int(s)
        print("Conversion succeeded! x =", x)
    except ValueError:
        print("Conversion failed!")
        x = -1
    except TypeError:
        print("Conversion failed!")
        x = -1
    return x
```

# Manejo de excepciones

```
def convert(s):
    '''Convert to an integer.'''
    x = -1
    try:
        x = int(s)
        print("Conversion succeeded! x =", x)
    except (ValueError, TypeError):
        print("Conversion failed!")
    return x
```



# Manejo de excepciones

```
def convert(s):
    '''Convert to an integer.'''
    x = -1
    try:
        x = int(s)
    except (ValueError, TypeError):
        return x
```



# Tipos de errores del programador

IndentationError

SyntaxError

NameError

# Manejo de excepciones

```
def convert(s):
    '''Convert to an integer.'''
    x = -1
    try:
        x = int(s)
    except (ValueError, TypeError):
        pass
    return x
```



# Manejo de excepciones

```
'''A module for demonstrating exceptions.'''\n\nimport sys\n\ndef convert(s):\n    '''Convert to an integer.'''\n    try:\n        return int(s)\n    except (ValueError, TypeError) as e:\n        print("Conversion error: {}\n".format(str(e)),\n              file=sys.stderr)\n    return -1
```



## Errores imprudentes

```
from math import log

def string_log(s):
    v = convert(s)
    return log(v)
```

## Re-raising Exceptions

```
def convert(s):
    '''Convert to an integer.'''
    try:
        return int(s)
    except (ValueError, TypeError) as e:
        print("Conversion error: {}".format(str(e)),
              file=sys.stderr)
        +
    raise
```



# Las excepciones son parte del API

El que llama a una función debe saber que excepción esperar y en qué momento

# roots.py

```
def sqrt(x):
    '''Compute square roots using the method of Heron of Alexandria.

    Args:
        x: The number for which the square root is to be computed.

    Returns:
        The square root of x.
    '''
    guess = x
    i = 0
    while guess * guess != x and i < 20:
        guess = (guess + x / guess) / 2.0
        i += 1
    return guess

def main():
    print(sqrt(9))
    print(sqrt(2))

if __name__ == '__main__':
    main()
```

roots.py



# roots.py

Usar excepciones que los usuarios puedan anticipar

Standard Exceptions son a menudo la mejor opción

# roots.py

```
def sqrt(x):
    '''Compute square roots using the method of Heron of Alexandria.

    Args:
        x: The number for which the square root is to be computed.

    Returns:
        The square root of x.
    '''

    guess = x
    i = 0
    while guess * guess != x and i < 20:
        guess = (guess + x / guess) / 2.0
        i += 1
    return guess

def main():
    try:
        print(sqrt(9))
        print(sqrt(2))
        print(sqrt(-1))
        print("This is never printed.") ←
    except ZeroDivisionError:
        print("Cannot compute square root of a negative number.")

    print("Program execution continues normally here.")

if __name__ == '__main__':
    main()
```



# Manejo de excepciones

Las excepciones son parte de la familia de las funciones relacionadas más conocidas como “protocolos”

Usar siempre excepciones comunes o existentes cuando sea posible

IndexError KeyError ValueError etc...

IndexError: cuando el índice no existe en una lista

KeyError: Cuando una llave no existe en un diccionario, cuando un mapeo falla

ValueError: Cuando el objeto es del tipo correcto pero tiene un valor inapropiado

# Manejo de excepciones

Evita proteger los programas sobre errores  
de tipo TypeError

usualmente eso no vale la pena

puede limitar las funciones  
innecesariamente

```
def convert(s):
    '''Convert to an integer.'''
    if not isinstance(s, str):
        raise TypeError(
            "Argument must be a string")

try:
    return int(s)
except (ValueError, TypeError) as e:
    print("Conversion error: {}".format(str(e)),
          file=sys.stderr)
    raise
```

float?

Fraction?

complex?

etc.



## Manejo de excepciones

Look Before You Leap

vs.

It's Easier to Ask Forgiveness  
than Permission

# Manejo de excepciones

It's **Easier to Ask Forgiveness**  
than **Permission**



LBYL

```
import os

p = '/path/to/datafile.dat'

if os.path.exists(p):
    process_file(p)
else:
    print('No such file as {}'.format(p))
```

## LBYL

```
import os

p = '/path/to/datafile.dat'

if os.path.exists(p):
    process_file(p)
else:
    print('No such file as {}'.format(p))
```

*Race condition*

# EAFP

```
p = '/path/to/datafile.dat'

try:
    process_file(f)
except OSError as e:
    print('Could not process file because{}'\
          .format(str(e)))
```



# Códigos de Error vs Excepciones

Los códigos de error son silenciosos por defecto

Códigos de error + LBYL

Las excepciones necesitan ser explícitamente manejadas

EAFP + Excepciones

Los errores son muy difícil de ignorar

# Acciones de Limpieza

```
import os

def make_at(path, dir_name):
    original_path = os.getcwd()
    os.chdir(path)
    os.mkdir(dir_name)
    os.chdir(original_path)
```

If this fails...



...then this won't happen!

+

# Acciones de Limpieza

```
import os

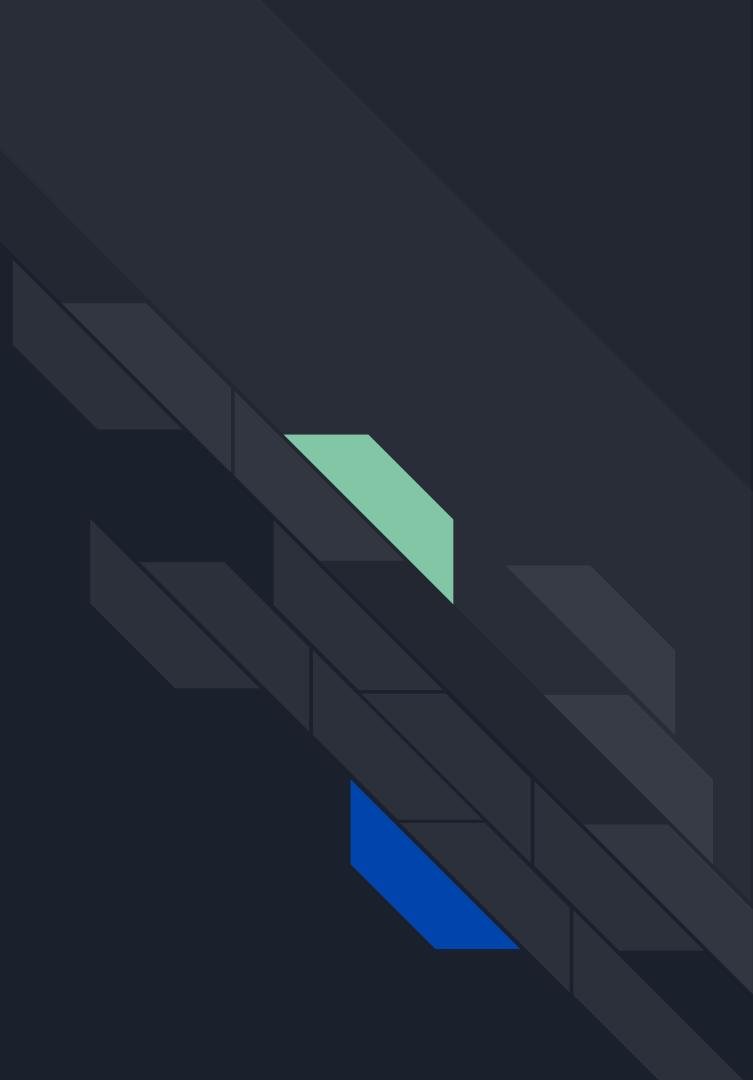
def make_at(path, dir_name):
    original_path = os.getcwd()
    try:
        os.chdir(path)
        os.mkdir(dir_name)      +
    finally:
        os.chdir(original_path)
```

# Acciones de Limpieza

```
import os
import sys

def make_at(path, dir_name):
    original_path = os.getcwd()
    try:
        os.chdir(path)
        os.mkdir(dir_name)
    except OSError as e:
        print(e, file=sys.stderr)
        raise
    finally:
        os.chdir(original_path)
```

Errors should never pass  
silently.  
Unless explicitly  
silenced.





# Módulos Específicos por plataforma

Windows:

- msvcrt

Linux

- sys

Unix

- tty

MacOS

- termios

# Módulos Específicos por plataforma

```
"""keypress - A module for detecting a single keypress."""

try:
    import msvcrt

    def getkey():
        """Wait for a keypress and return a single character string."""
        return msvcrt.getch()

except ImportError:

    import sys
    import tty
    import termios

    def getkey():
        """Wait for a keypress and return a single character string."""
        fd = sys.stdin.fileno()
        original_attributes = termios.tcgetattr(fd)
        try:
            tty.setraw(sys.stdin.fileno())
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, original_attributes)
        return ch

    # If either of the Unix-specific tty or termios are not found,
    # we allow the ImportError to propagate from here
```



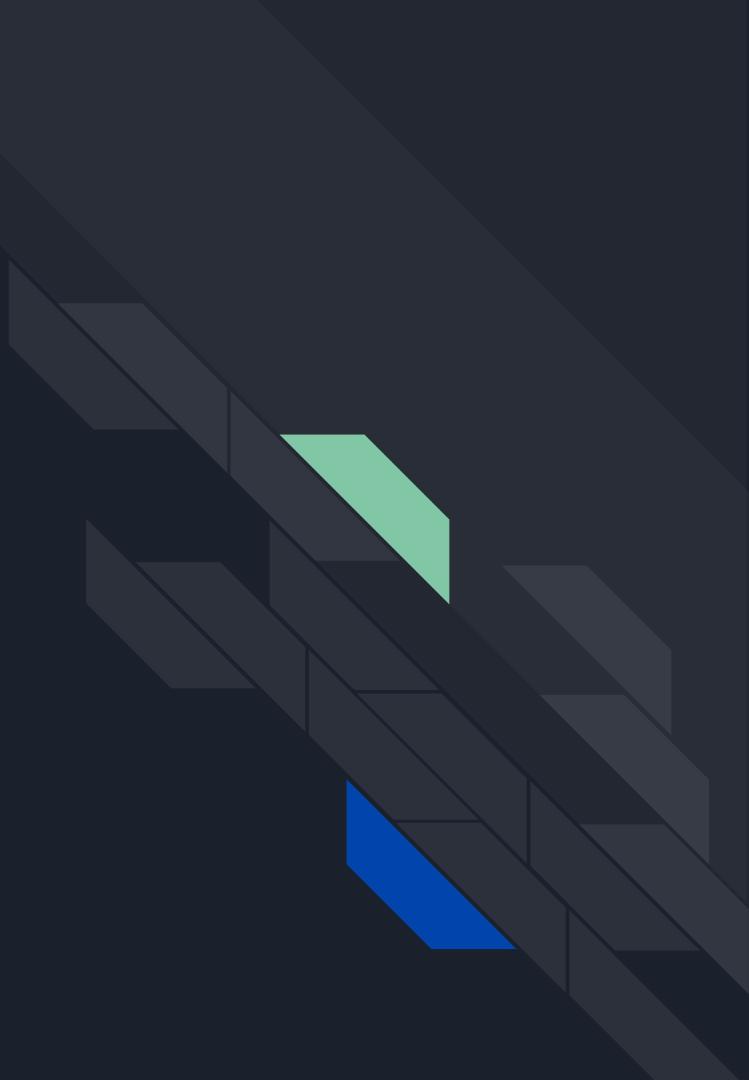
# Tarjeta de Embarque

Desarrollar una función que reciba los datos de:

Número de Vuelo, Aerolínea, Origen, Destino, Número de Asiento, Nombre, Apellido, Edad del Ocupante y que lo imprima con un banner hecho en consola con el nombre de la empresa vendedora de boleto

Validar todos los campos y lanzar todas las excepciones necesarias. Documentar

# Comprehensiones Avanzado





# Comprehensiones

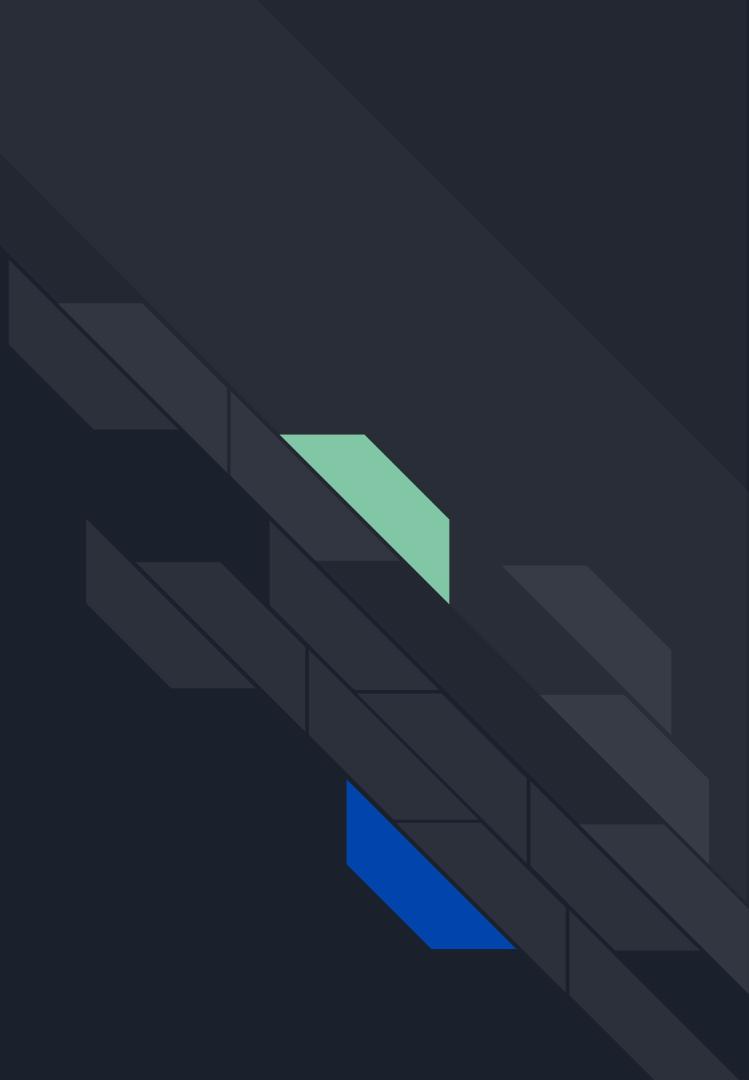
Permiten denotar en un formato declarativo o funcional a

- Listas
- SET
- Diccionarios

Las comprensiones son: legibles, expresivas y efectivas

```
[expr(item) for item in iterable] { expr(item) for item in iterable } (expr(item) for item in iterable)  
{ key_expr:value_expr for item in iterable }  
[ expr(item) for item in iterable if predicate(item) ]
```

Simple is better than  
complex.





# Generadores

una de las características más elegantes y poderosas de Python

- Especifican secuencias iterables
  - todos los generadores son iteradores
- son evaluadas de forma retrasada
  - el valor siguiente es calculado por demanda
- permite modelar secuencias infinitas
  - Como streams de datos con un final no definido (archivos de log, datos de sensores)
- Pueden estar compuestos por pipeline
  - permite tratar el stream de datos de forma natural

```
def gen123():
    yield 1
    yield 2
    yield 3
```

- Por lo menos debe existir la palabra clave yeild 1 vez
- Puede tener un return sin parámetros

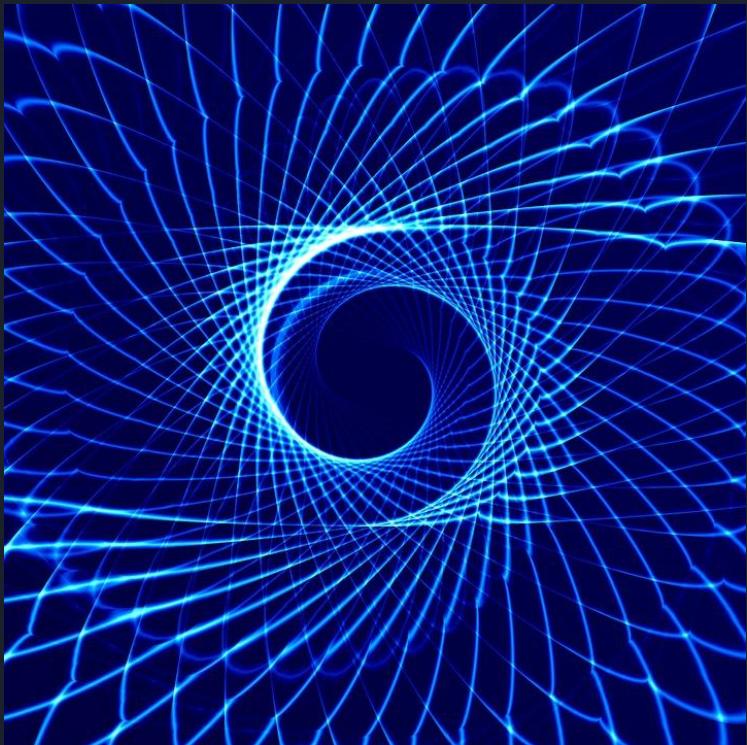
# stateful generators

- Se puede resumir la ejecución
- Pueden mantener el estado en variables locales
- Pueden tener un control de flujo complejo
- Evaluación con retardo

```
def take(count, iterable):
    "Take first count elements"
    counter = 0
    for item in iterable:
        if counter == count:
            return
        counter += 1
        yield item
```

# Lazy generators

- Se usa para manejo de archivos gigantes ( terabytes)
- Recuperar información sobre sensores (IoT)
- Manejo de flujo de datos infinitos
- Evaluación al llegar el dato





# comprehensiones generadores

- Sintaxis igual a comprensiones pero se usa ()
- Crean un objeto generador
- Concisos
- Evaluación con retardo
- de un solo uso
- Se puede usar los () de las funciones

( expr(item) **for** item **in** iterable)

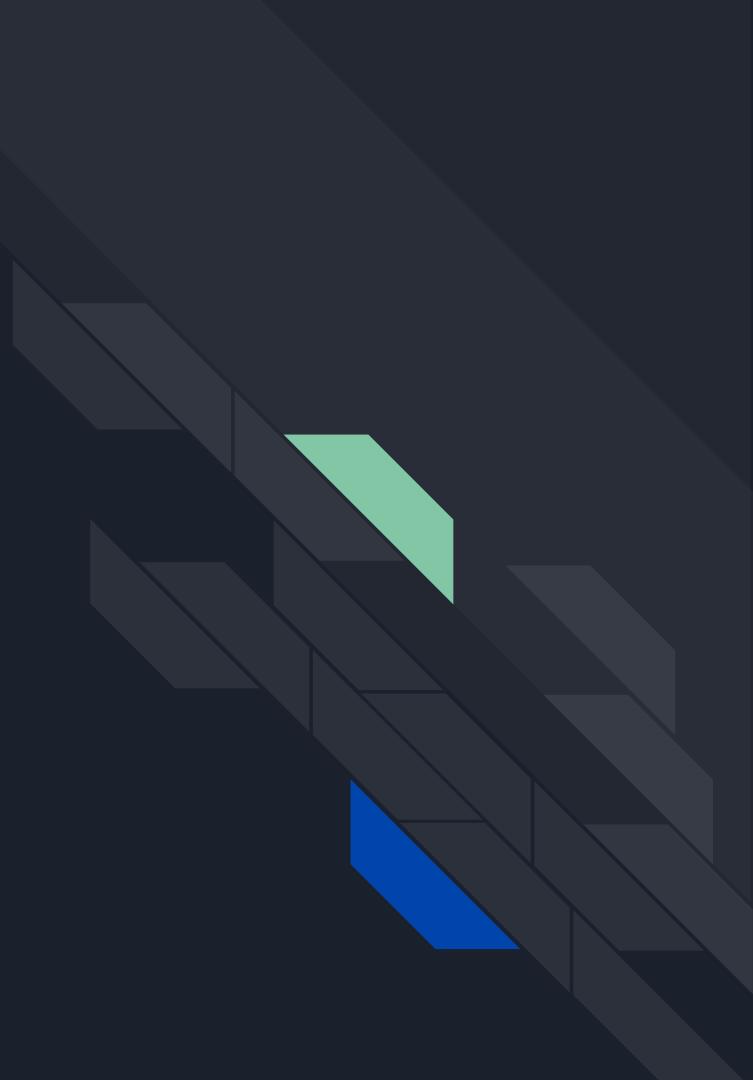
( expr(item) **for** item **in** iterable **if** expr(item) )



# Práctica

Escribir una función "diaSiguienteT" que dada una fecha expresada (Día, Mes, Año) (donde Día y Año son números enteros, y Mes es el texto "Ene", "Feb", ..., "Dic", según corresponda) calcule el día siguiente al dado, en el mismo formato. Adicional deberá imprimir el mes entero de la fecha solicitada. Deberá validar cualquier error de ingreso de datos

# Clases en Python





# Clases

- Todo en python es una clase
  - Clases son herramientas que permiten:
    - Crear nuevos tipos de datos
    - Crear nuevas funcionalidades
  - Las clases en Python proveen toda la funcionalidad estándar de Programación orientada a objetos
  - Las clases soportan dos tipos de operaciones:
    - instanciación
    - referencia de atributo
  - Para nombrar las clases se usa CapsWords o CamelCase PEP 8
- Las clases definen la estructura y el comportamiento de un objeto
- Un objeto clase controla su inicialización
- Las clases permite que un problema complejo sea manejable
- Las clases permite entregar soluciones simples



# Instancia y métodos

- Método es una función definida por la clase
- la instancia de un método pueden ser llamado por objetos
- Self es el primer argumento para todas las instancias de métodos
- `_init_()` es método de instancia que permite inicializar nuevos objetos
- `_init_()` es un inicializador no un constructor
- self funciona como this en java o c++
- En Python el constructor es proveído por sistema de runtime
  - revisa que exista el inicializador y lo llama cuando está presente
  - El constructor se llama `_new_` que normalmente retorna un objeto válido pero des poblado
  - Comúnmente (nunca) se hace una sobreescritura de `_init_` y `_new_` a la vez



# Por Qué `_number` ?

No es necesario crear un atributo del objeto antes de crearlo. y debido a esto se utiliza el `_`

- Esto permite que el nombre del atributo no choque con el nombre del método
- Por convención los detalles de los objetos que no van a ser manipulados o consumidos usan como prefijo un underscore `_`
- Ver PEP 8 - Descriptive: Naming Styles

<https://www.python.org/dev/peps/pep-0008/#id36>

# Public Private Protected

- En Python todo es público, no existe Privado ni Protegido.
- Por cultura
- Python deja esa simulación de seguridad y alienta a los programadores a ser responsables.
- En la práctica, esto funciona muy bien.





# class invariants

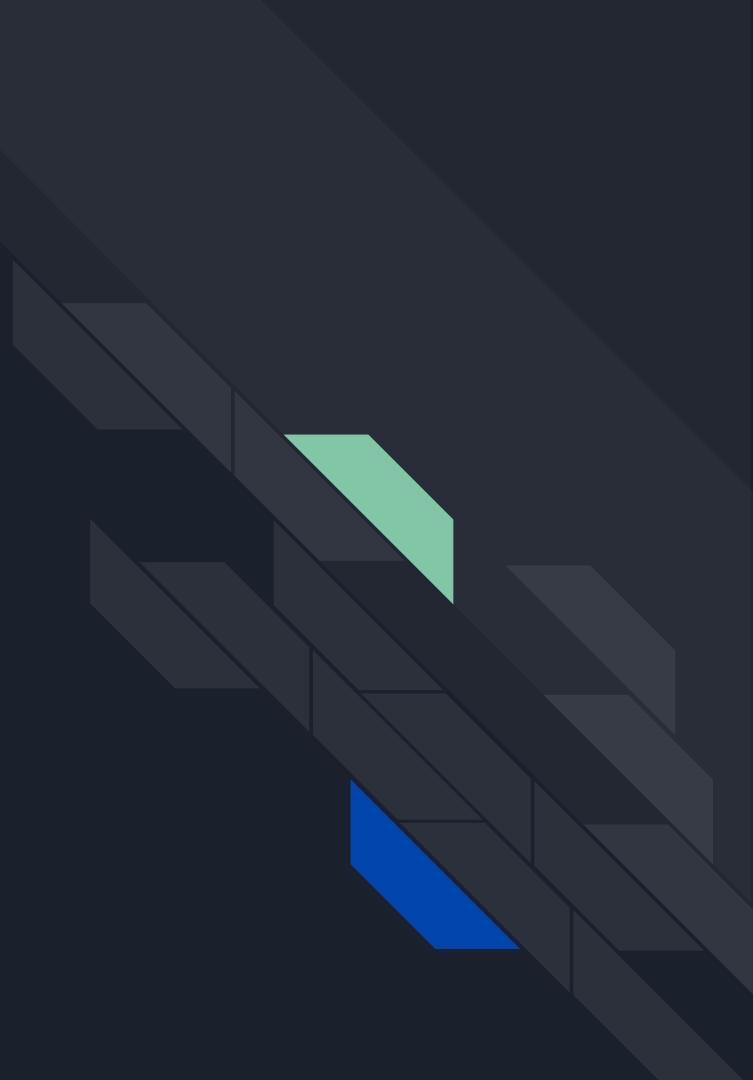
- Es una buena práctica para el inicializador de un objeto establecer los invariantes de clase
- los invariantes son verdades sobre los objetos de esa clase que deben perdurar por la vida del objeto.
- Ejemplo: En Número de un vuelo los primeros dos caracteres pertenecen al código de la línea aérea seguido por 3 o 4 caracteres que es el número de ruta



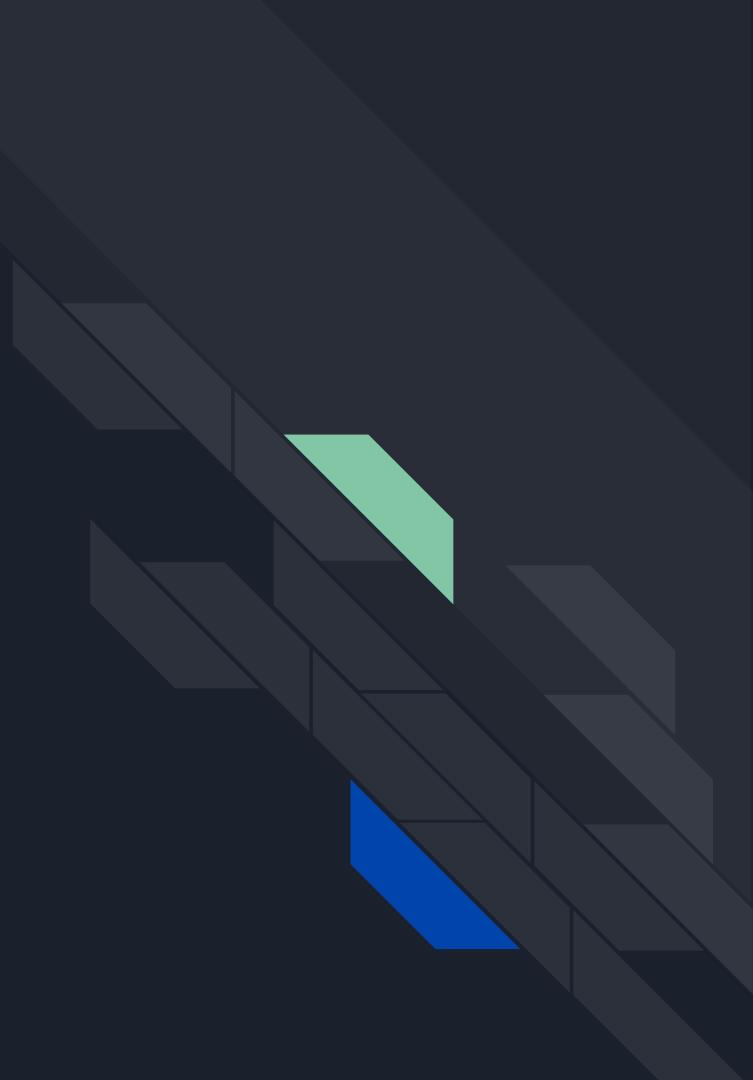
# Clases Colaborativas

- Se puede definir varias clases en un mismo archivo
- Ley de Dimitrio - principio de diseño
  - El principio del último conocido
  - Solo habla con tus amigos
- Se debe documentar las clases.  
docstring
- Se puede instanciar una clase en el inicializador de otra clase

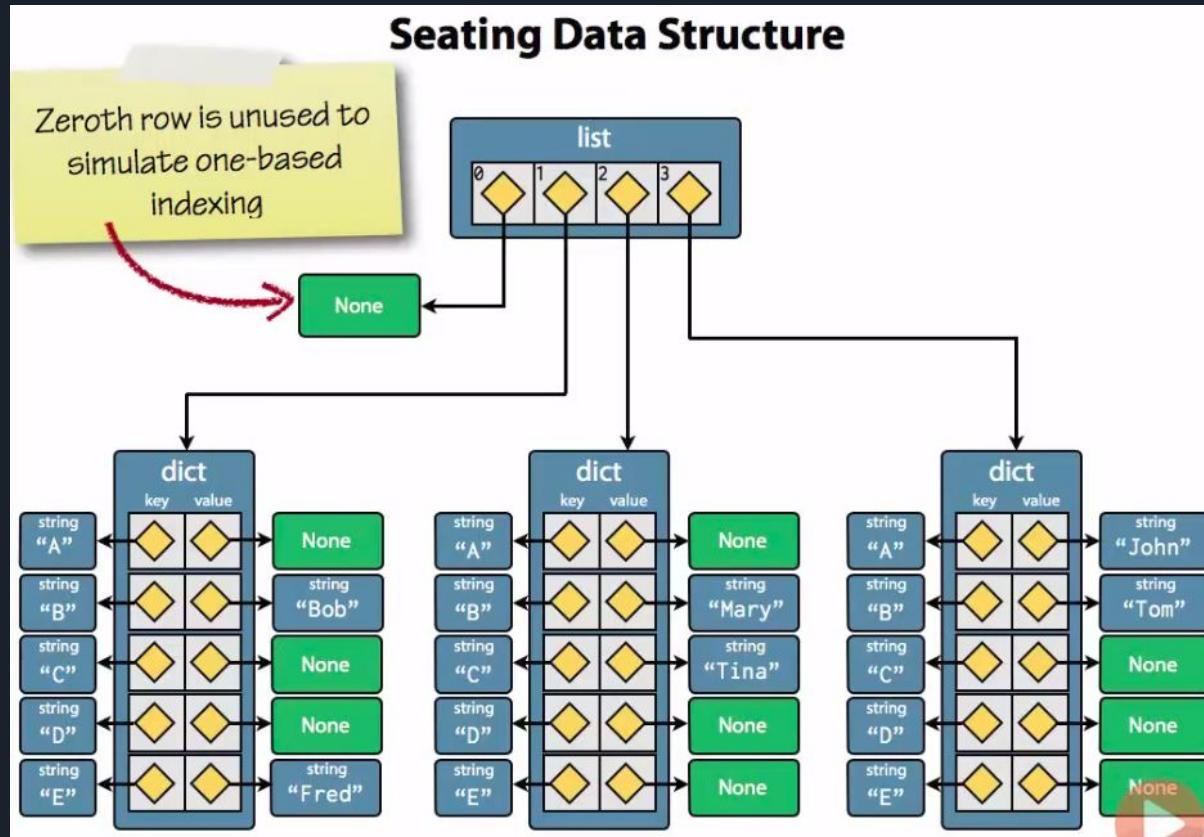
Complex is better than  
complicated



# Ejemplo: Reserva de Asientos



# Reserva de Asientos



# Código de reserva

```
filas, asientos = self._aeronave.plan_asientos()  
self._asientos = [None] + [{letra: None for letra in asientos} for _ in filas]
```

# Asignar asientos

```
def asignar_asiento(self, asiento, pasajero):
    filas, letras_asientos = self._aeronave.plan_asientos()

    letra = asiento[-1]
    if letra not in letras_asientos:
        raise ValueError("Letra de asiento invalida {}".format(letra))

    fila_texto = asiento[:-1]
    try:
        fila = int(fila_texto)
    except ValueError:
        raise ValueError("Fila de asiento invalida {}".format(fila_texto))

    if fila not in filas:
        raise ValueError("Numero de fila invalida {}".format(fila))

    if self._asientos[fila][letra] is not None:
        raise ValueError("El asiento {} esta ocupado".format(asiento))

    self._asientos[fila][letra] = pasajero
```

# re asignar pasajeros

```
32     # Como esto es un detalle se usa
33     def _parsear_asientos(self, asiento):
34         filas, letras_asientos = self._aeronave.plan_asientos()
35
36         letra = asiento[-1]
37         if letra not in letras_asientos:
38             raise ValueError("Letra de asiento invalida {}".format(letra))
39
40         fila_texto = asiento[:-1]
41         try:
42             fila = int(fila_texto)
43         except ValueError:
44             raise ValueError("Fila de asiento invalida {}".format(fila_texto))
45
46         if fila not in filas:
47             raise ValueError("Numero de fila invalida {}".format(fila))
48
49         return fila, letra
50
51     def asignar_asiento(self, asiento, pasajero):
52         fila, letra = self._parsear_asientos(asiento)
53
54         if self._asientos[fila][letra] is not None:
55             raise ValueError("El asiento {} esta ocupado".format(asiento))
56
57         self._asientos[fila][letra] = pasajero
58
59     def reasignar_pasajero(self, asiento_origen, asiento_destino):
60         fila_origen, letra_origen = self._parsear_asientos(asiento_origen)
61         if self._asientos[fila_origen][letra_origen] is None:
62             raise ValueError("No hay Pasajero en el asiento {}".format(asiento_origen))
63
64         fila_destino, letra_destino = self._parsear_asientos(asiento_destino)
65         if self._asientos[fila_destino][letra_destino] is not None:
66             raise ValueError("El asiento {} está ocupado".format(asiento_destino))
67
68         self._asientos[fila_destino][letra_destino] = self._asientos[fila_origen][asiento_origen]
69         self._asientos[fila_origen][letra_origen] = None
```

# instanciar

```
def crear_vuelo():
    f = Vuelo("AB2455", Aeronave(234, "Boing 747", 13, 4))
    f.asignar_asiento("5A", "Antonio")
    f.asignar_asiento("10D", "Andres")
    f.asignar_asiento("12B", "Paola")
    f.asignar_asiento("8C", "Santiago")
    return f
```

# Imprimir cantidad de asientos libres

```
def numero_asientos_libres(self):
    return sum(sum(1 for s in filas.values() if s is None)
              for filas in self._asientos
              if filas is not None)
```

# Tarjeta de Embarque - OO con funciones

```
def _asientos_pasajeros(self):
    numeros_fila, letras_asiento = self._aeronave.plan_asientos()
    for filas in numeros_fila:
        for letras in letras_asiento:
            pasajero = self._asientos[filas][letras]
            if pasajero is not None:
                yield(pasajero, '{}{}'.format(filas, letras))

def crear_tarjeta_embarque(self, impresion_tarjeta):
    for pasajero, asiento in sorted(self._asientos_pasajeros()):
        impresion_tarjeta(pasajero, asiento, self.numero(), self.modelo_nave()) +  
+  
def tarjeta_embarque_consola(pasajero, asiento, numero_vuelo, Aeronave):
    salida = "| Nombre {0}" \
            " Vuelo {1}" \
            " Asiento {2}" \
            " Aeronave {3}" \
            "|".format(pasajero, numero_vuelo, asiento, Aeronave)
    banner = '+' + '-' * (len(salida) - 2) + '+'
    borde = '[' + ' ' * (len(salida) - 2) + ']'
    linea = [banner, borde, salida, borde, banner]
    tarjeta = '\n'.join(linea)
    print(tarjeta)
    print()
```



# Polimorfismo

- usar objetos de diferentes tipos a través de una misma interfaz
  - En el ejemplo anterior crear\_tarjeta\_embarque es polimórfica ya que se puede pasar una función que imprima en html5

## Duck Typing

“Cuando veo una ave, que camina como pato, nada como pato, y grazna como un pato, a esa ave la llamo pato”

- James Whitcomb Riley

# Polimorfismo

```
class AirbusA319:  
    def __init__(self, registro):  
        self._registro = registro  
  
    def registro(self):  
        return self._registro  
  
    def modelo(self):  
        return "Airbus A319"  
  
    def plan_asientos(self):  
        return range(1, 23), "ABCDEF"  
  
class Boeing747:  
    def __init__(self, registro):  
        self._registro = registro  
  
    def registro(self):  
        return self._registro  
  
    def modelo(self):  
        return "Boeing 747"  
  
    def plan_asientos(self):  
        return range(1, 57), "ABCDEFGH"
```

```
def crear_vuelo():  
    f = Vuelo("AB2455", AirbusA319("WS-123"))  
    f.asignar_asiento("5A", "Antonio")  
    f.asignar_asiento("10D", "Andres")  
    f.asignar_asiento("12B", "Paola")  
    f.asignar_asiento("8C", "Santiago")  
  
    g = Vuelo("AG2432", Boeing747("GS-345"))  
    g.asignar_asiento("1A", "Guillermo")  
    g.asignar_asiento("14G", "Susana")  
    g.asignar_asiento("10A", "Diana")  
    g.asignar_asiento("10B", "Jhonatan")  
    return f, g
```

# Herencia

Una sub-clase se puede derivar de una super-clase

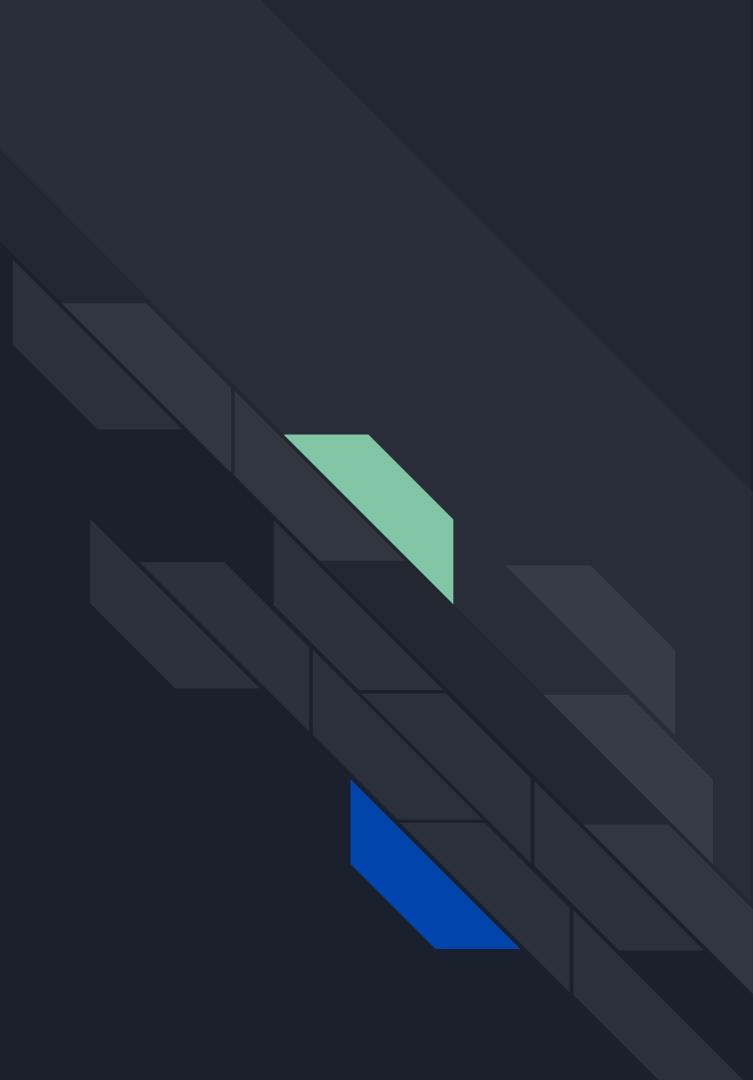
la herencia es un mecanismo por el cual podemos crear comportamientos más específicos

En Python la herencia es más usada para compartir implementaciones

```
class Aeronave:  
    # TODO en produccion todo deberia ser validado  
    def __init__(self, registro):  
        self._registro = registro  
  
    def registro(self):  
        return self._registro  
  
    def numero_asientos(self):  
        filas, filas_asientos = self.plan_asientos()  
        return len(filas), len(filas_asientos)  
  
class AirbusA319(Aeronave):  
  
    def modelo(self):  
        return "Airbus A319"  
  
    def plan_asientos(self):  
        return range(1, 23), "ABCDEF"  
  
class Boeing747(Aeronave):  
  
    def modelo(self):  
        return "Boeing 747"  
  
    def plan_asientos(self):  
        return range(1, 57), "ABCDEFGH"
```



# Manejo de Archivos





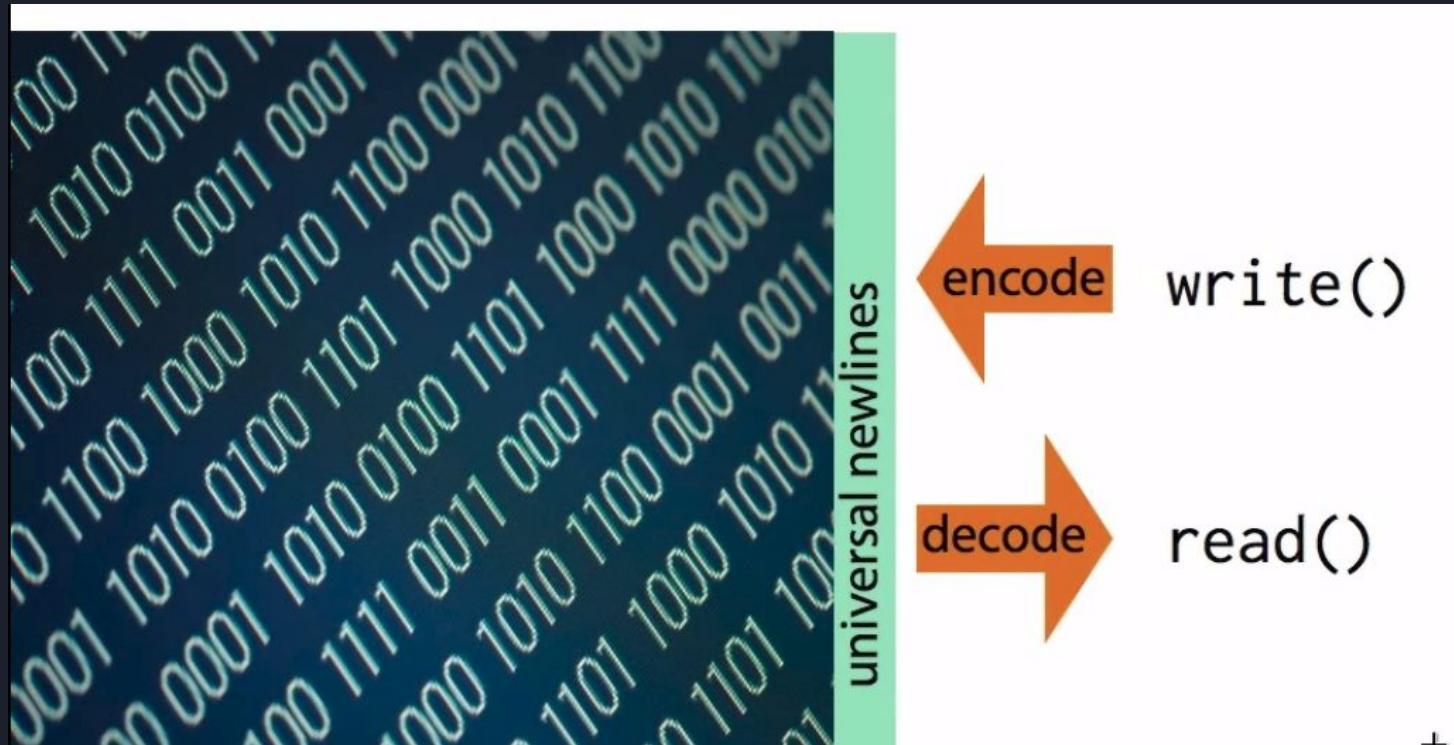
# Open()

Función Built-in:

Abre un archivo

- File: ruta y nombre del archivo (requerido)
- Mode: Modo en el cual se abrirá el archivo, debe existir permisos en el FS.
  - read/write/append
  - binary/text
- Encoding: Codificación usada
  - <https://docs.python.org/3/library/codecs.html>

# Acceso a archivos de texto



# Open Modes

## open() modes

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)
'U'	universal newlines mode (for backwards compatibility; should not be used in new code)



# write()

Retorna el número de caracteres escritos no el tamaño.

modo: wt

el archivo se modifica al realizar el .close()

El desarrollador debe manejar los saltos de línea



# read()

permite leer un archivo

modo: rt

seek()

read()

readable()

readline



append()

writelines()

# Archivos como iteradores

```
import sys

def main(filename):
    f = open(filename, mode='rt', encoding='utf-8')
    for line in f:
        sys.stdout.write(line)
    f.close()

if __name__ == "__main__":
    main(sys.argv[1])
```

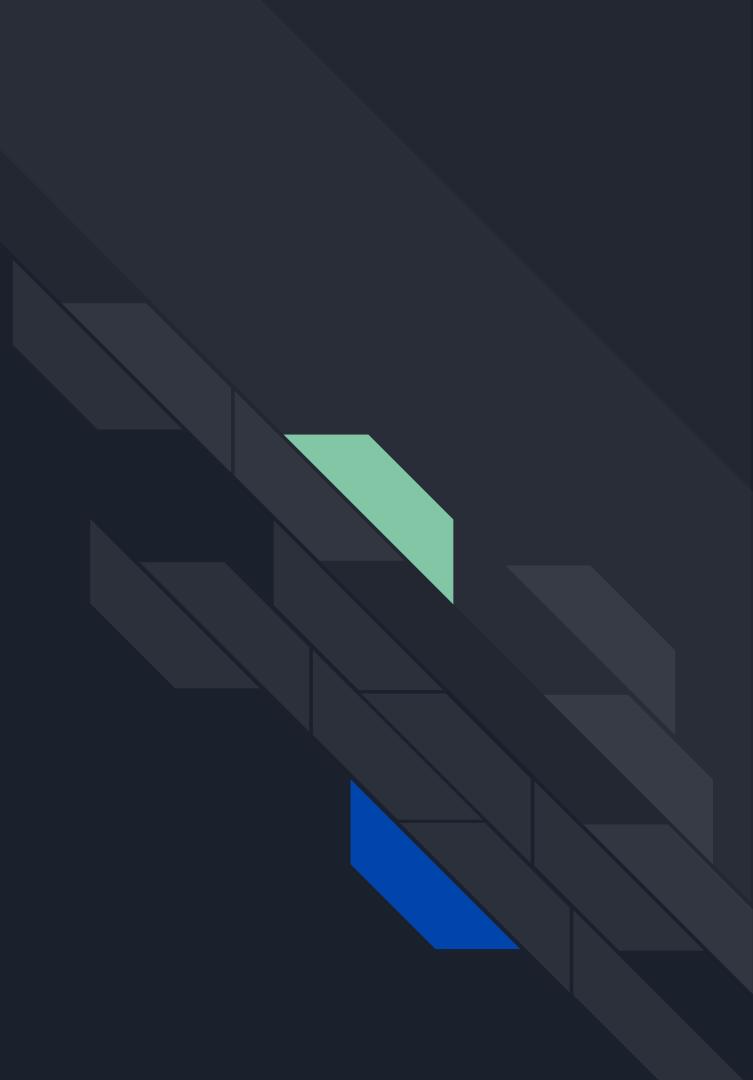




# Manejo de Archivos con try..finally

```
def read_series(filename):
    try:
        f = open(filename, mode='rt', encoding='utf-8')
        return [int(line.strip()) for line in f]
    finally:
        f.close()
```

# Context Manager - with block





## Uso típico de un archivo

```
f = open()  
# work work work  
f.close()
```



# with block



open() returns a  
context-manager!

# with-block

resource cleanup with context-managers

+



with block

```
with EXPR as VAR:  
    BLOCK
```



# with block

```
def read_series(filename):
    with open(filename, mode='rt', encoding='utf-8') as f:
        return [int(line.strip()) for line in f]
```



# Publicando, Trabajando y Manteniendo código

UnitTest, Packaging, Debugging

# Unit Test



# unittest

unit  
tests

integration  
tests

acceptance  
tests

+

# Unit Test

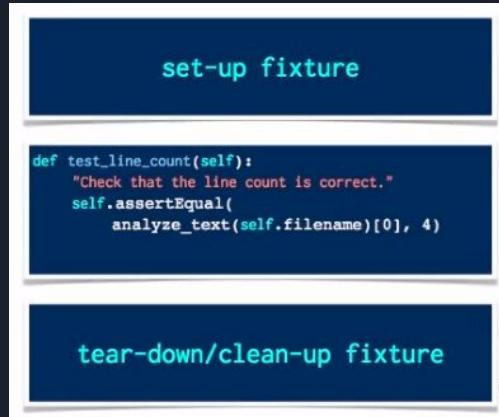


# unittest

automated  
&  
repeatable

# Unit Test

- Test Case: Agrupa funciones de pruebas relacionadas ( unidad básica de pruebas)
- Fixtures: código que se ejecuta antes y/o después de cada función de prueba, los Fixture se utilizan para configurar un estado esperado para que las pruebas puedan ejecutarse. (como por ejemplo una tabla populada), también se utilizan para limpiar cualquier objeto utilizado durante la prueba



# Unit Test

- assertions: pruebas específicas para condiciones y comportamientos. las aserciones son las que deciden si una prueba ha sido superada o fallida. adicional pueden realizar:
  - Pruebas booleanas simples
  - Pruebas de igualdad de objetos
  - Verificar que se lancen las excepciones adecuadas

x.is\_valid()

x == y

raise ValueError()

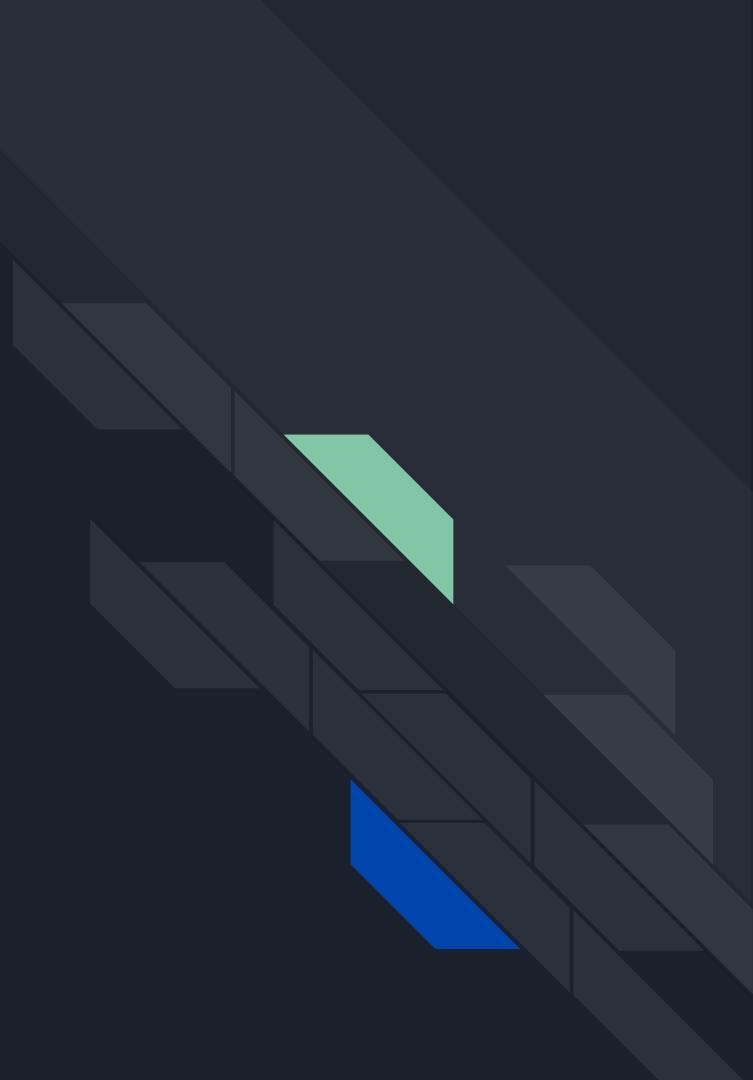
If an assertion  
fails, then a test  
fails.

TDD

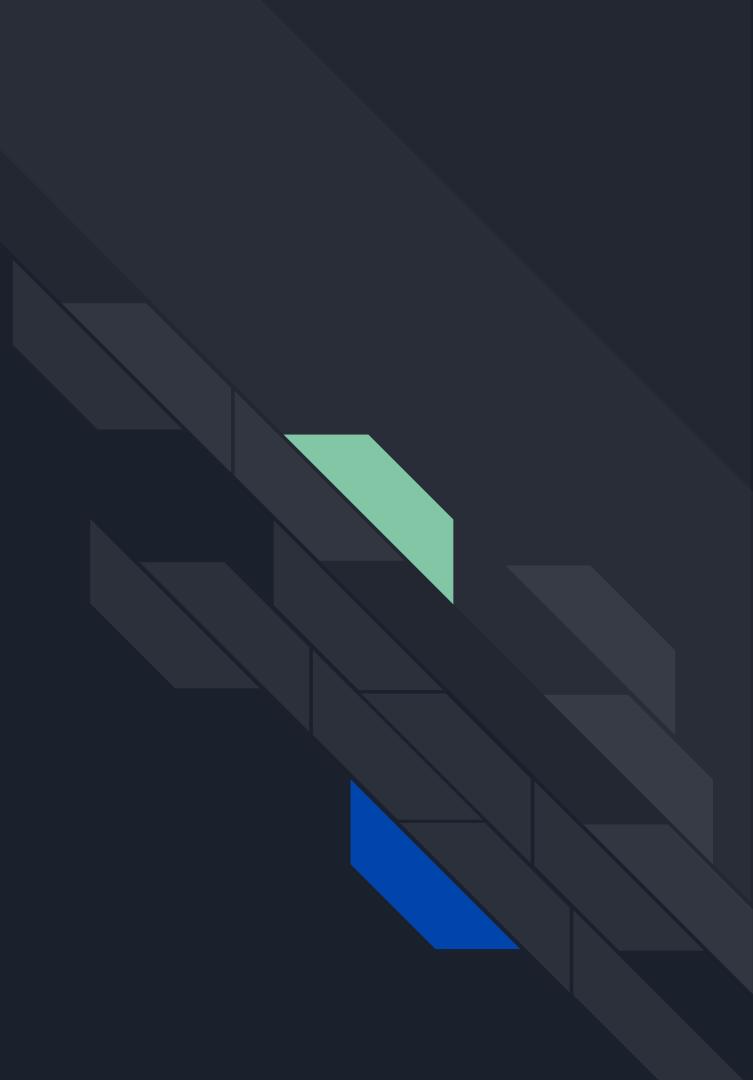
# Test-Driven Development



# Demo Unit Test



# Debug

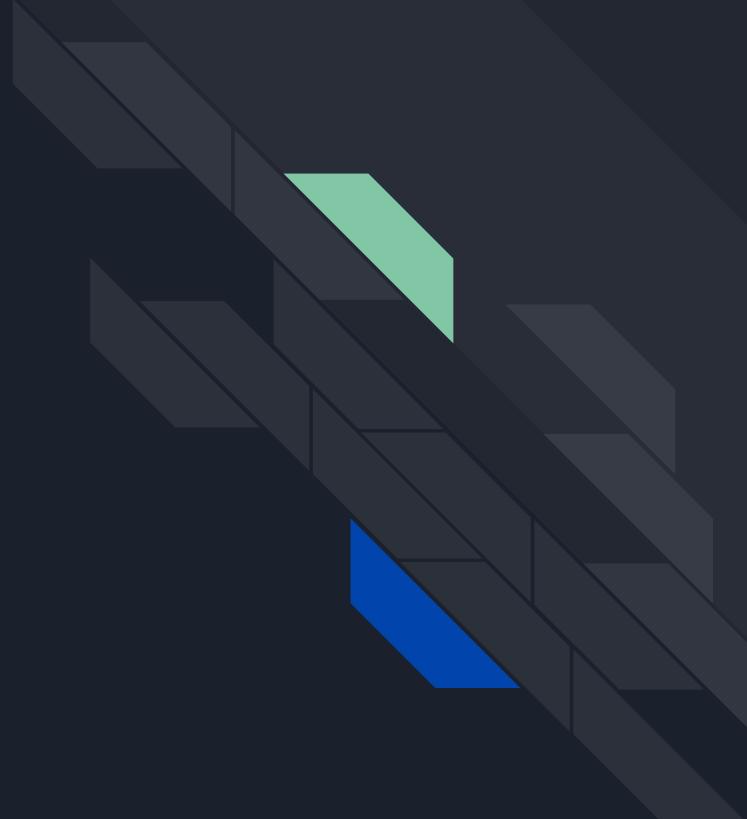


# Debug

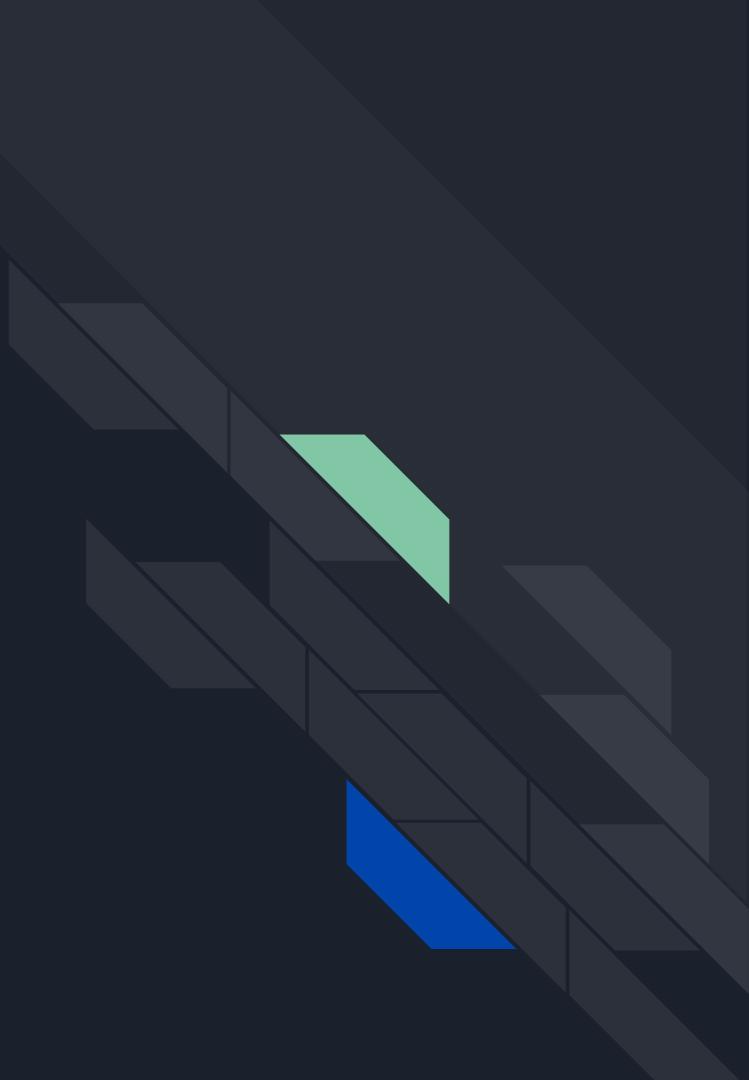
- Es necesario para saber qué pasa con un método, función, o variable
- Puede revisar contenidos, valores
- Existen varias herramientas tanto de consola como GUI.
  - VS code usa PTVSD
  - PDB se puede usar por comando en el caso que no se tenga un GUI, pero es tedioso, es built-in



# DEMO Debugging



# Flask





Qué es Flask?



- Flask Is based on the Werkzeug WSGI toolkit AND Jinja2 template engine.



# Getting started with Python & Flask

Install Python

```
brew install python
```

Install Virtual Environment

```
# pip install virtualenv
```

Make a new virtual environment

```
# virtualenv venv
```

Activite the virtual environment and setup tools

```
# source venv/bin/activate
```



## Install Flask and its dependencies

```
# pip install flask
```

### Test flask installation

- Make the Hello.py

```
# vim Hello.py
```

- And paste in

```
#!/usr/bin/env python
```

```
from flask import Flask  
app = Flask(__name__)
```

```
@app.route('/')  
def hello_world()  
    return 'Hello World'  
if __name__ == '__main__':  
    app.run()
```



Save, exit and run the script

```
# python Hello.py
```

Open a browser and go to <http://127.0.0.1.5000>

## run() method of flask class `app.run(host, port, debug, options)`

Sr.No	Parameters & Description
1	<b>host</b> Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to '0.0.0.0' to have server available externally
2	<b>port</b> Defaults to 5000
3	<b>debug</b> Defaults to false. If set to true, provides a debug information
4	<b>options</b> To be forwarded to underlying Werkzeug server.

## Debug mode

```
app.debug = True  
app.run()  
app.run(debug= True)
```

# Flask – Routing

Routing technique, `route()` decorator is used to bind URL to a function

```
@app.route('/hello')
def hello_world():
    return 'hello world'
```

```
def hello_world():
    return 'hello world'
app.add_url_rule('/', 'hello', hello_world)
```

# Flask – Variable Rules

keyword argument to the function → <variable-name>

```
from flask import Flask
app = Flask(__name__)

@app.route('/hello/<name>')
def hello_name(name):
    return 'Hello %s!' % name

if __name__ == '__main__':
    app.run(debug = True)
```

```
from flask import Flask
app = Flask(__name__)

@app.route('/blog/<int:postID>')
def show_blog(postID):
    return 'Blog Number %d' % postID

@app.route('/rev/<float:revNo>')
def revision(revNo):
    return 'Revision Number %f' % revNo

if __name__ == '__main__':
    app.run()
```

Sr.No	Converters & Description
1	<b>int</b> accepts integer
2	<b>float</b> For floating point value
3	<b>path</b> accepts slashes used as directory separator character

# Flask – URL Building

## url\_for() function

```
from flask import Flask, redirect, url_for
app = Flask(__name__)

@app.route('/admin')
def hello_admin():
    return 'Hello Admin'

@app.route('/guest/<guest>')
def hello_guest(guest):
    return 'Hello %s as Guest' % guest

@app.route('/user/<name>')
def hello_user(name):
    if name =='admin':
        return redirect(url_for('hello_admin'))
    else:
        return redirect(url_for('hello_guest',guest = name))

if __name__ == '__main__':
    app.run(debug = True)
```

# Flask – HTTP methods

Sr.No	Methods & Description
1	<b>GET</b> Sends data in unencrypted form to the server. Most common method.
2	<b>HEAD</b> Same as GET, but without response body
3	<b>POST</b> Used to send HTML form data to server. Data received by POST method is not cached by server.
4	<b>PUT</b> Replaces all current representations of the target resource with the uploaded content.
5	<b>DELETE</b> Removes all current representations of the target resource given by a URL

# Flask – HTTP methods

```
<html>
  <body>

    <form action = "http://localhost:5000/login" method = "post">
      <p>Enter Name:</p>
      <p><input type = "text" name = "nm" /></p>
      <p><input type = "submit" value = "submit" /></p>
    </form>

  </body>
</html>
```

```
from flask import Flask, redirect, url_for, request
app = Flask(__name__)

@app.route('/success/<name>')
def success(name):
    return 'welcome %s' % name

@app.route('/login',methods = [ 'POST', 'GET'])
def login():
    if request.method == 'POST':
        user = request.form['nm']
        return redirect(url_for('success',name = user))
    else:
        user = request.args.get('nm')
        return redirect(url_for('success',name = user))

if __name__ == '__main__':
    app.run(debug = True)
```

# Flask – Templates

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return '<html><body><h1>Hello World</h1></body></html>'

if __name__ == '__main__':
    app.run(debug = True)
```

- ❑ Application folder
  - ❑ Hello.py
  - ❑ templates
    - ❑ hello.html

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('hello.html')

if __name__ == '__main__':
    app.run(debug = True)
```

# Web Templating System

```
<!doctype html>
<html>
  <body>

    <h1>Hello {{ name }}!</h1>

  </body>
</html>
```

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/hello/<user>')
def hello_name(user):
    return render_template('hello.html', name = user)

if __name__ == '__main__':
    app.run(debug = True)
```

# Flask – Static Files

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")

if __name__ == '__main__':
    app.run(debug = True)
```

index.html



```
<html>

    <head>
        <script type = "text/javascript"
            src = "{{ url_for('static', filename = 'hello.js') }}" ></script>
    </head>

    <body>
        <input type = "button" onclick = "sayHello()" value = "Say Hello" />
    </body>

</html>
```

hello.js →

```
function sayHello() {
    alert("Hello World")
}
```



# Flask – Request Object

- **Form** – It is a dictionary object containing key and value pairs of form parameters and their values.
- **args** – parsed contents of query string which is part of URL after question mark (?).
- **Cookies** – dictionary object holding Cookie names and values.
- **files** – data pertaining to uploaded file.
- **method** – current request method.

# Flask – Sending Form Data to Template

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route('/')
def student():
    return render_template('student.html')

@app.route('/result',methods = ['POST', 'GET'])
def result():
    if request.method == 'POST':
        result = request.form
        return render_template("result.html",result = result)

if __name__ == '__main__':
    app.run(debug = True)
```

# Flask – Sending Form Data to Template

```
<html>
  <body>

    <form action = "http://localhost:5000/result" method = "POST">
      <p>Name <input type = "text" name = "Name" /></p>
      <p>Physics <input type = "text" name = "Physics" /></p>
      <p>Chemistry <input type = "text" name = "chemistry" /></p>
      <p>Maths <input type ="text" name = "Mathematics" /></p>
      <p><input type = "submit" value = "submit" /></p>
    </form>

  </body>
</html>
```

# Flask – Sending Form Data to Template

```
<!doctype html>
<html>
    <body>

        <table border = 1>
            {% for key, value in result.items() %}

                <tr>
                    <th> {{ key }} </th>
                    <td> {{ value }} </td>
                </tr>

            {% endfor %}
        </table>

    </body>
</html>
```

# Flask – Cookies

- Request object

```
make_response()  
set_cookie()
```

- The `get()` method of `request.cookies` attribute is used to read a cookie.

```
@app.route('/')  
def index():  
    return render_template('index.html')
```

```
<html>  
  <body>  
  
    <form action = "/setcookie" method = "POST">  
      <p><h3>Enter userID</h3></p>  
      <p><input type = 'text' name = 'nm'/></p>  
      <p><input type = 'submit' value = 'Login' /></p>  
    </form>  
  
  </body>  
</html>
```



# Flask – Cookies

```
@app.route('/setcookie', methods = ['POST', 'GET'])
def setcookie():
    if request.method == 'POST':
        user = request.form['nm']

        resp = make_response(render_template('readcookie.html'))
        resp.set_cookie('userID', user)

    return resp
```



# Flask – Cookies

```
@app.route('/setcookie', methods = ['POST', 'GET'])
def setcookie():
    if request.method == 'POST':
        user = request.form['nm']

        resp = make_response(render_template('readcookie.html'))
        resp.set_cookie('userID', user)

    return resp
```

# Flask – Sessions

```
@app.route('/')
def index():
    if 'username' in session:
        username = session['username']
        return 'Logged in as ' + username + '<br>' + \
            "<b><a href = '/logout'>click here to log out</a></b>"
    return "You are not logged in <br><a href = '/login'></b>" + \
        "<b><a href = '/login'>click here to log in</a></b>"
```

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))
    return '''

        <form action = "" method = "post">
            <p><input type = text name = username/></p>
            <p><input type = submit value = Login/></p>
        </form>

    '''
```

# Flask – Sessions

`logout()` view function

```
@app.route('/logout')
def logout():
    # remove the username from the session if it is there
    session.pop('username', None)
    return redirect(url_for('index'))
```

set `secret_key`

```
from flask import Flask, session, redirect, url_for, escape, request
app = Flask(__name__)
app.secret_key = 'any random string'
```

# Flask – Redirect & Errors

Flask.redirect(location, statuscode, response)

- **location** parameter is the URL where response should be redirected.
- **statuscode** sent to browser's header, defaults to 302.
- **response** parameter is used to instantiate response.

status codes



- HTTP\_300\_MULTIPLE\_CHOICES
- HTTP\_301\_MOVED\_PERMANENTLY
- HTTP\_302\_FOUND
- HTTP\_303\_SEE\_OTHER
- HTTP\_304\_NOT\_MODIFIED
- HTTP\_305\_USE\_PROXY
- HTTP\_306\_RESERVED
- HTTP\_307\_TEMPORARY\_REDIRECT

# Flask – Redirect & Errors

```
from flask import Flask, redirect, url_for, render_template, request
# Initialize the Flask application
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('log_in.html')

@app.route('/login',methods = ['POST', 'GET'])
def login():
    if request.method == 'POST' and
    request.form['username'] == 'admin' :
        return redirect(url_for('success'))
    return redirect(url_for('index'))

@app.route('/success')
def success():
    return 'logged in successfully'

if __name__ == '__main__':
    app.run(debug = True)
```



# Flask – Redirect & Errors

Flask.abort(code)

- **400** – for Bad Request
- **401** – for Unauthenticated
- **403** – for Forbidden
- **404** – for Not Found
- **406** – for Not Acceptable
- **415** – for Unsupported Media Type
- **429** – Too Many Requests

# Flask – Redirect & Errors

```
from flask import Flask, redirect, url_for, render_template, request, abort
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('log_in.html')

@app.route('/login', methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        if request.form['username'] == 'admin' :
            return redirect(url_for('success'))
        else:
            abort(401)
    else:
        return redirect(url_for('index'))

@app.route('/success')
def success():
    return 'logged in successfully'

if __name__ == '__main__':
    app.run(debug = True)
```



# Flask – Message Flashing

`flash(message, category)`

- **message** parameter is the actual message to be flashed.
- **category** parameter is optional. It can be either 'error', 'info' or 'warning'.

# Flask – Message Flashing

```
get_flashed_messages(with_categories, category_filter)
```

```
{% with messages = get_flashed_messages() %}
  {% if messages %}
    {% for message in messages %}
      {{ message }}
    {% endfor %}
  {% endif %}
{% endwith %}
```

```
@app.route('/')
def index():
    return render_template('index.html')
```

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
    error = None

    if request.method == 'POST':
        if request.form['username'] != 'admin' or \
           request.form['password'] != 'admin':
            error = 'Invalid username or password. Please try again!'
    else:
        flash('You were successfully logged in')
        return redirect(url_for('index'))
    return render_template('login.html', error = error)
```

# Flask – Message Flashing

Login.html →

```
<!doctype html>
<html>
  <body>

    <h1>Login</h1>

    {% if error %}
      <p><strong>Error:</strong> {{ error }}</p>
    {% endif %}

    <form action = "" method = post>
      <dl>
        <dt>Username:</dt>
        <dd>
          <input type = text name = username
                 value = "{{request.form.username }}">
        </dd>

        <dt>Password:</dt>
        <dd><input type = password name = password></dd>
      </dl>
      <p><input type = submit value = Login></p>
    </form>

  </body>
</html>
```

# Flask – Message Flashing

Index.html ➔

```
<!doctype html>
<html>

    <head>
        <title>Flask Message flashing</title>
    </head>
    <body>

        {% with messages = get_flashed_messages() %}
            {% if messages %}
                <ul>
                    {% for message in messages %}
                        <li>{{ message }}</li>
                    {% endfor %}
                </ul>
            {% endif %}
        {% endwith %}

        <h1>Flask Message Flashing Example</h1>
        <p>Do you want to <a href = "{{ url_for('login') }}">
            <b>log in?</b></a></p>

    </body>
</html>
```

# Flask – Message Flashing

Flash.py ➔

```
from flask import Flask, flash, redirect, render_template, request, url_for
app = Flask(__name__)
app.secret_key = 'random string'

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/login', methods = ['GET', 'POST'])
def login():
    error = None

    if request.method == 'POST':
        if request.form['username'] != 'admin' or \
           request.form['password'] != 'admin':
            error = 'Invalid username or password. Please try again!'
        else:
            flash('You were successfully logged in')
            return redirect(url_for('index'))

    return render_template('login.html', error = error)

if __name__ == "__main__":
    app.run(debug = True)
```



# Flask – File Uploading

- `request.files[file]`
- `secure_filename()` function

Configuration settings of Flask object

<code>app.config['UPLOAD_FOLDER']</code>	Defines path for upload folder
<code>app.config ['MAX_CONTENT_PATH']</code>	Specifies maximum size of file to be uploaded – in bytes

# Flask – File Uploading

‘upload.html’ →

```
<html>
  <body>

    <form action = "http://localhost:5000/uploader" method = "POST"
      enctype = "multipart/form-data">
      <input type = "file" name = "file" />
      <input type = "submit"/>
    </form>

  </body>
</html>
```

# Flask – File Uploading

**Submit** → Form's post method invokes '/upload\_file' URL

```
from flask import Flask, render_template, request
from werkzeug import secure_filename
app = Flask(__name__)

@app.route('/upload')
def upload_file():
    return render_template('upload.html')

@app.route('/uploader', methods = ['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['file']
        f.save(secure_filename(f.filename))
        return 'file uploaded successfully'

if __name__ == '__main__':
    app.run(debug = True)
```

# Flask – Extensions

- **Flask Mail** – provides SMTP interface to Flask application
- **Flask WTF** – adds rendering and validation of WTForms
- **Flask SQLAlchemy** – adds SQLAlchemy support to Flask application
- **Flask Sijax** – Interface for Sijax - Python/jQuery library that makes AJAX easy to use in web applications

```
from flask_foo import [class, function]
```

```
from flask.ext import foo → For versions of Flask later than 0.7 ( compatibility module needs  
to be activated)
```

```
import flaskext_compat  
flaskext_compat.activate()  
from flask.ext import foo
```

# Flask – Mail

- Install

```
pip install Flask-Mail
```

## Configuration settings

Sr.No	Parameters & Description
1	<b>MAIL_SERVER</b> Name/IP address of email server
2	<b>MAIL_PORT</b> Port number of server used
3	<b>MAIL_USE_TLS</b> Enable/disable Transport Security Layer encryption
4	<b>MAIL_USE_SSL</b> Enable/disable Secure Sockets Layer encryption
5	<b>MAIL_DEBUG</b> Debug support. Default is Flask application's debug status

6	<b>MAIL_USERNAME</b> User name of sender
7	<b>MAIL_PASSWORD</b> password of sender
8	<b>MAIL_DEFAULT_SENDER</b> sets default sender
9	<b>MAIL_MAX_EMAILS</b> Sets maximum mails to be sent
10	<b>MAIL_SUPPRESS_SEND</b> Sending suppressed if app.testing set to true
11	<b>MAIL_ASCII_ATTACHMENTS</b> If set to true, attached filenames converted to ASCII

# Flask – WTF

- Install Flask-WTF extension  
`pip install flask-WTF`

## Standard form fields:

Sr.No	Standard Form Fields & Description	
1	<b>TextField</b> Represents <code>&lt;input type = 'text'&gt;</code> HTML form element	5 <b>RadioField</b> Represents <code>&lt;input type = 'radio'&gt;</code> HTML form element
2	<b>BooleanField</b> Represents <code>&lt;input type = 'checkbox'&gt;</code> HTML form element	6 <b>SelectField</b> Represents select form element
3	<b>DecimalField</b> Textfield for displaying number with decimals	7 <b>TextAreaField</b> Represents <code>&lt;testarea&gt;</code> html form element
4	<b>IntegerField</b> TextField for displaying integer	8 <b>PasswordField</b> Represents <code>&lt;input type = 'password'&gt;</code> HTML form element
		9 <b>SubmitField</b> Represents <code>&lt;input type = 'submit'&gt;</code> form element



# Flask – SQLAlchemy

*What is ORM (Object Relation Mapping)?*

- Install **Flask-SQLAlchemy extension**.  
`pip install flask-sqlalchemy`
- Import **SQLAlchemy class from this module**.  
`from flask_sqlalchemy import SQLAlchemy`
- Create a **Flask application object and set URI for the database to be used**.

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///students.sqlite3'
```