

# *Python - Generalidades*

## **Autor:**

Juan David Argüello Plata - Ingeniero Mecánico

## **Profesor tutor:**

Jairo René Martínez Morales - Químico PhD

CENIVAM

Universidad Industrial de Santander

# Introducción

## ¿Qué es Python?

Python es:

- Libre.
- Multipropósito.
- Dinámico.
- Fácil de aprender y usar.

## ¿Qué puedo llegar a hacer con Python?

- Páginas web.
- Aplicaciones móviles.
- Sistemas que empleen **Inteligencia Artificial y Aprendizaje Automático** (machine learning).
- Automatización de sistemas mediante Arduino y Raspberry Pi.
- Automatización de tareas (ejemplo en Jupyter).

# Objetivo de hoy

Hoy veremos:

- Naturaleza de las variables.
  - ▶ Números.
  - ▶ Cadenas de texto.
  - ▶ Listas.
  - ▶ Tuplas.
  - ▶ Diccionarios.
- Depuración: try - except.

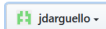
# Repositorio - Git (opcional)

Para **practicar**, crea un repositorio en tu cuenta personal en GitHub para que *guardes* tus avances.

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

Owner



Repository name \*

Great repository names are short and memorable. Need inspiration? How about **fluffy-funicular**?

Description (optional)



**Public**

Anyone can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

Add .gitignore: **None**

Add a license: **None**



Create repository



# Lectura del código

El código de programación se ejecuta **siempre** desde arriba hacia abajo.


```
import numpy as np
import itertools as it
import matplotlib.pyplot as plt
from scipy.stats import norm
from matplotlib.ticker import PercentFormatter

class NormalDist():
    """
        Datos para la gráfica de distribución normal y
        diagrama de pareto
    """
    def __init__(self, data):
        self.NormG = {}
        comb = self.Combinaciones(data)
        self.CombMatrix(comb, data)
        self.Orden()
        self.FractionZ()

    def FractionZ(self):
        nT = 0
        for key in self.NormG:
            nT += 1
        for key in self.NormG:
            self.NormG[key]['Fracción'] = (self.NormG[key]['Orden']-0.5)/nT
            self.NormG[key]['Z'] = norm.ppf(self.NormG[key]['Fracción'])
            self.NormG['N_Datos'] = len(self.NormG[key]['Vector'])

    def Orden(self):
        datos = [[], []]
        for var in self.NormG:
            datos[0].append(self.NormG[var]['X'])
        datos[1] = np.searchsorted(np.sort(datos[0]), datos[0], side='left')

        for var in self.NormG:
            for i in range(len(datos[0])):
                if datos[0][i] == self.NormG[var]['X']:
                    self.NormG[var]['Orden'] = datos[1][i]
```



# *Variables en Python*

Empieza **creando** un “Script”(archivo de extensión .py que contiene el algoritmo de programación). Abre tu editor de texto de preferencia (en mi caso, Sublime Text), y guarda el archivo.

## *Guardar código*

<Nombre del archivo> + '.py'

Crea una variable **numérica**:

## *Crear variable (numérica)*

```
x = 2
```

Para imprimir...

## *Imprime!*

```
print(x)
```

# Operaciones con Variables

Ejecuta el algoritmo.

*Pulsa:*

CTRL + B

Las operaciones básicas...

*Matemática básica...*

$$y = (2 * (x + 2) ** (3 * x)) / 5$$

Que equivale a...

$$y = \frac{2 (x + 2)^{3x}}{5} \quad (1)$$

## *Variables - Cadenas de texto*

Podemos crear variables de texto con sólo escribir:

*Texto...*

```
frase = "Hola, puedo escribir lo que quiera"
```

O incluso, agregar *más* texto...

*¿Seguro?*

```
print(frase + '... ¿Estás seguro?')
```

El texto es, en realidad, **un vector**:

*Miremos letras...*

```
print(frase[0])
```

Última letra.

*Más letras...*

```
print(frase[len(frase)-1])
```



## *Variables - Listas*

Son vectores *dinámicos*, lo que quiere decir que pueden **cambiar** sus valores a través del algoritmo de programación.

Creemos una lista.

*Crear lista*

```
l = [1, 'Hola', False, 0]
```

Podemos cambiar un valor.

*¿Terminamos?*

```
l[1] = 'Chao'
```

*¿De verdad?*

*Incrédulo*

```
l[2] = True
```

## *Variables - Tuplas*

Son vectores *estáticos*: **no** cambian sus componentes durante la ejecución del código.

Creación de una tupla...

*Tupla*

```
t = (1, 'Hola', 0)
```

¿Cambia sus valores?

*No...*

```
t[1] = 'Chao'
```

## *Variables - Diccionario*

Tipo de variable dinámica que **facilita** la interpretación y el orden del código. Puede contener *cualquier* tipo de variable.

Crear un diccionario.

### *Crear Diccionario*

```
dic = {'Pal': 'Algo', 'Num': 2, 'Bool': True, 'List': [0,'a'], 'Tup': (0,1)}
```

Es modificable...

### *Modificar Num*

```
dic['Num'] += 1
```

# Depuración de código

La depuración consiste en **probar** el algoritmo lógico.

¿**Por qué?** Fácil: No nos las sabemos todas...

Por ejemplo:

*Código que falla...*

```
z = x/0
```

¿**Cómo podemos evitar que falle?**

- Sabiendo que ocurre antes del fallo con "print()".
- try - except.

## *Depuración de código: try - except*

Lo utilizamos cuando **no** nos interesa que falle ni el porqué falla. Nos interesa que continúe...

```
try :  
    z = x/0  
except :  
    pass
```

## *Depuración de código - Errores*

Es muy probable que nos encontremos con errores que **sí** nos interesa solucionar. *¿Qué podemos hacer?*

Podemos recurrir a [Stack Overflow](#) y buscar problemas *similares* de otros programadores, o **hacer** nuestras propias preguntas y esperar respuestas.