

A blue circuit board pattern is visible along the left edge of the slide.

FACULTAD DE  
INGENIERÍA DEL  
EJERCITO

UNIVERSIDAD DE LA  
DEFENSA NACIONAL

# PARADIGMAS DE PROGRAMACIÓN

V

# PRESENTACIÓN

- Profesores:
  - Nicolás Díaz País – [ndiazpais@fie.undef.edu.ar](mailto:ndiazpais@fie.undef.edu.ar)
    - Ing. En Informática (ITBA)
    - Docente Investigador – FIE
  - Liaus Pavel PAUL – [liaous\\_pavel@yahoo.com](mailto:liaous_pavel@yahoo.com)
    - Ing. En Sistemas (UTN)
    - CIDESO
- Consultas:
  - Por mail [ndiazpais@fie.undef.edu.ar](mailto:ndiazpais@fie.undef.edu.ar)
  - Por Slack
  - Por Campus Virtual -  
<https://campus.fie.undef.edu.ar/>



# BIBLIOGRAFÍA

- “Learn Python Programming” – Fabrizio Romano
- “Mastering Python” – Rick Van Hattem
- “The Hitchhiker’s guide to Python” – Reitz & Schlusser
- “Beginning Programming with Python” – John Paul Mueller
- IMPORTANTE: ASISTIR A CLASE!!

# ESQUEMA DE LA MATERIA

**Conceptos y claves del  
paradigma**

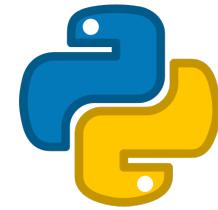


Modelado de objetos

Diseño de sistemas

Patrones de diseño y  
arquitectura

Aspectos Comerciales y  
Desarrollo de proyectos



**Implementaciones del  
paradigma**

Lenguaje Python 3.8+

Interfaces

- visuales
- protocolos
- bases de datos

# PUNTEROS



Base en la programación imperativa



Aritmética de Punteros



Memoria RAM y Memoria Virtual

# DEMO PUNTEROS 1 (ANSI C)

p =

0x7ffee518286c

0x7ffee5182870

var =

10

0x7ffee518286c



## DEMO PUNTEROS 2 (ANSI C)

`s[0] = 'P'`

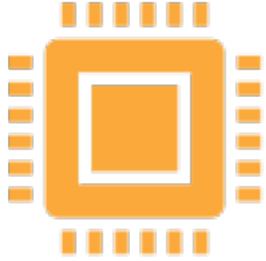
`s[1] = 'I'`

`s[2] = 'I'`

`s[3] = NULL (0x00)`

`s[4] = ???`

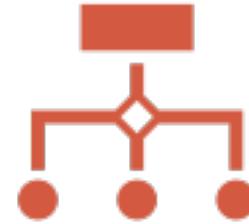
`s[0] + 4`



## Historia y Categorías

IBM a principios de los '70, mediante la técnica de *Time Sharing*.

Hoy en día, se virtualizan datacenters enteros!



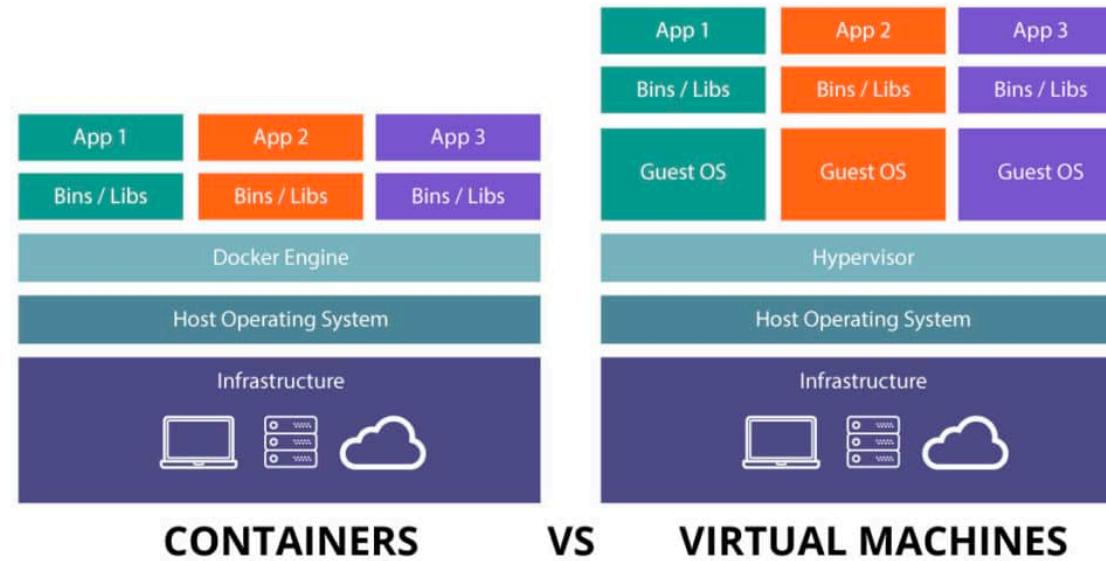
## Categorías de Virtualización

Sistemas: Sistemas operativos HOST y GUEST. Hypervisor. Full virtualization

Procesos: Pensados para ejecutar programas de forma independiente de la plataforma

Containers: Comparten el sistema HOST para aislar los diferentes servicios

# MAQUINAS VIRTUALES – HISTORIA Y CONCEPTOS



# COMPARACIÓN DE VIRTUALIZACIÓN

# MAQUINAS VIRTUALES – VENTAJAS Y CRITICAS



## Ventajas

Permite multiplataforma

Abstracciones del uso de periféricos (drivers) y bibliotecas

Mejor distribución del software

Solo importa la implementación para cada S.O.

¿Qué pasa con los punteros? Referencias dentro de la máquina virtual



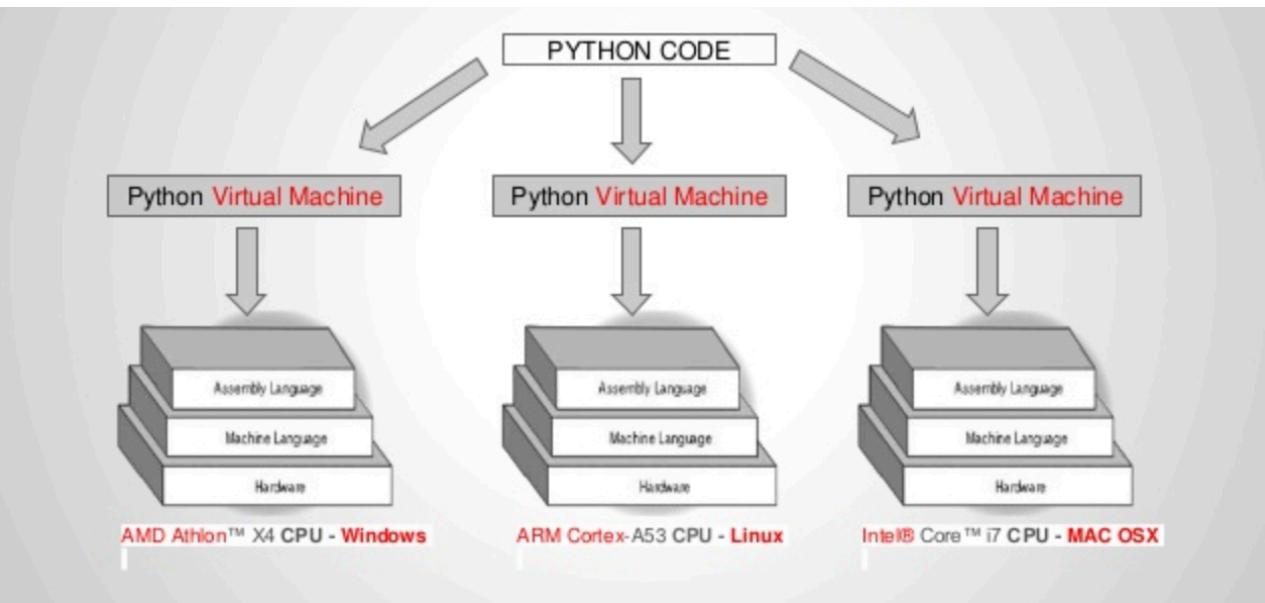
## Criticas

Agrega una capa de procesamiento.

No se ejecuta de forma “nativa”

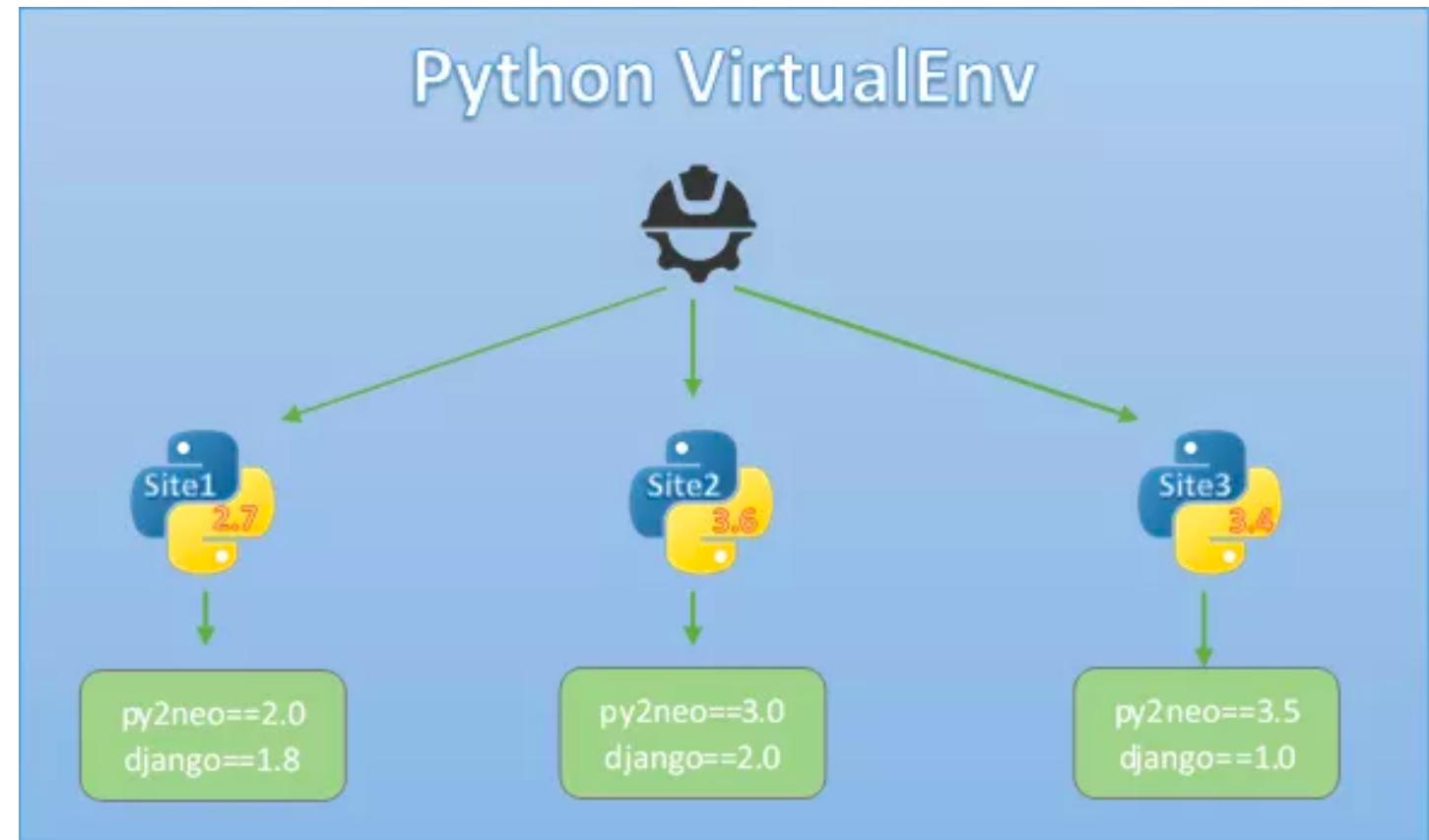
Potenciales fallas de seguridad de la implementación de cada VM

# PYTHON VIRTUAL MACHINE



# PYTHON VIRTUAL ENVIRONMENT

- Generalmente llamadas *VENVs*
- Permite no solo seleccionar la versión de Python a utilizar, sino la instalación de todos los paquetes necesarios de forma aislada
- Separación entre aplicaciones y dependencias
- No se trata de una virtualización completa



# CASOS DE ESTUDIO



Android APPs se ejecutan  
maquinas virtuales



Java: código fuente .java y  
.class (compiladas) con  
bytecode para VM



Java, Python, PHP deben  
gran parte de su éxito de  
popularidad por ser  
multiplataformas

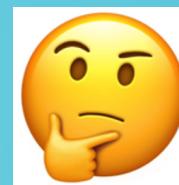


Python Virtual Machine  
¿interpretado o compilado?



Tanto JAVA como Python  
tienen incorporado el  
concepto de bytecode.

# BINDING ESTÁTICO

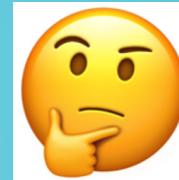


¿Qué pasa en C?

```
1 #include <stdio.h>
2
3 int main() {
4     int a;
5     int b;
6     char * c;
7
8     a = 10;
9     b = 20;
10
11    c = "P II";
12
13    printf("%d\n", a);
14    printf("%s\n", c);
15
16    a = c; ←
17
18    return 0;
19}
20
```

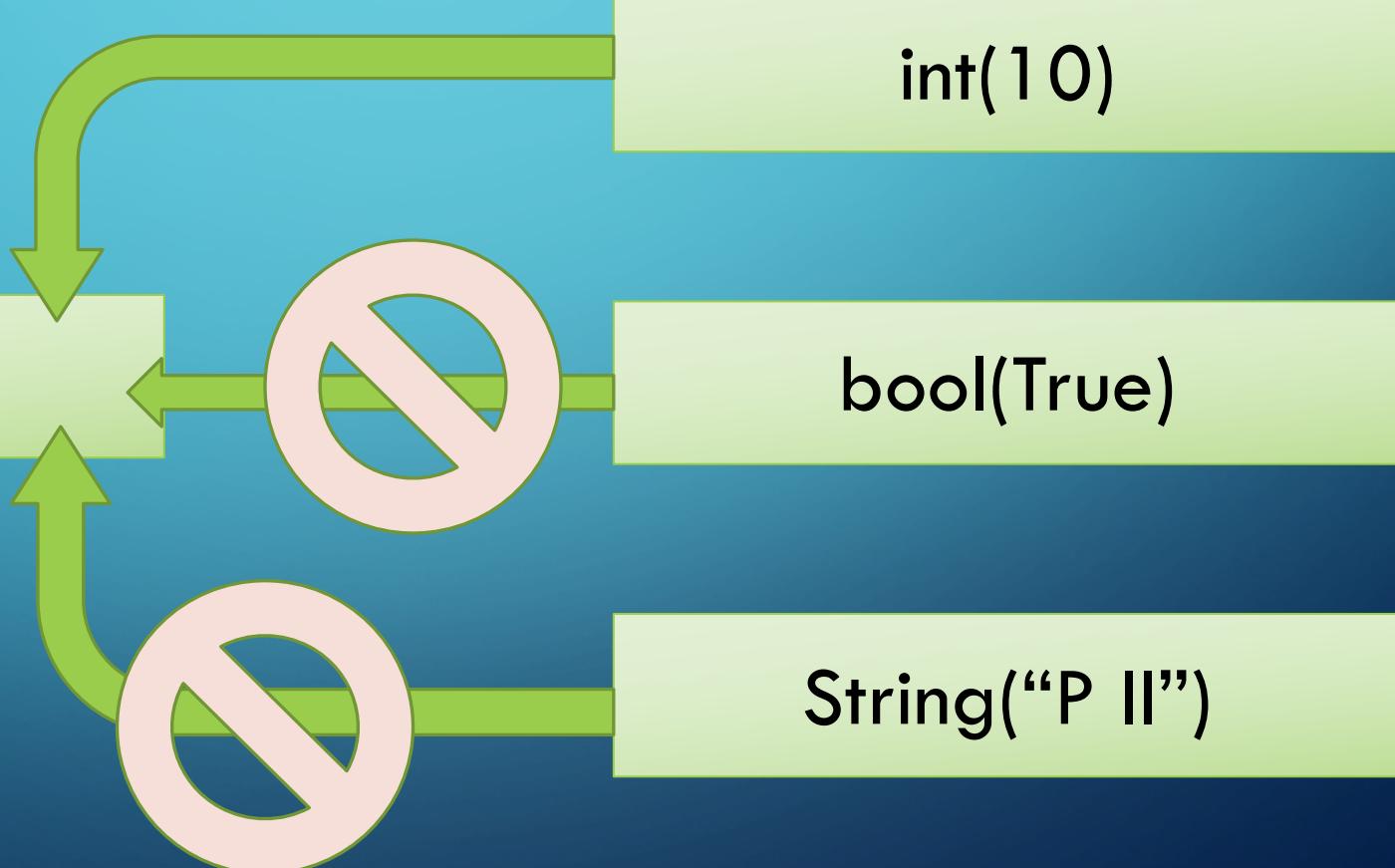
A thick red arrow points from the assignment statement `a = c;` to the declaration of `a` at line 4, indicating that the variable `a` is being reassigned to point to the string constant `"P II"`.

# BINDING ESTÁTICO



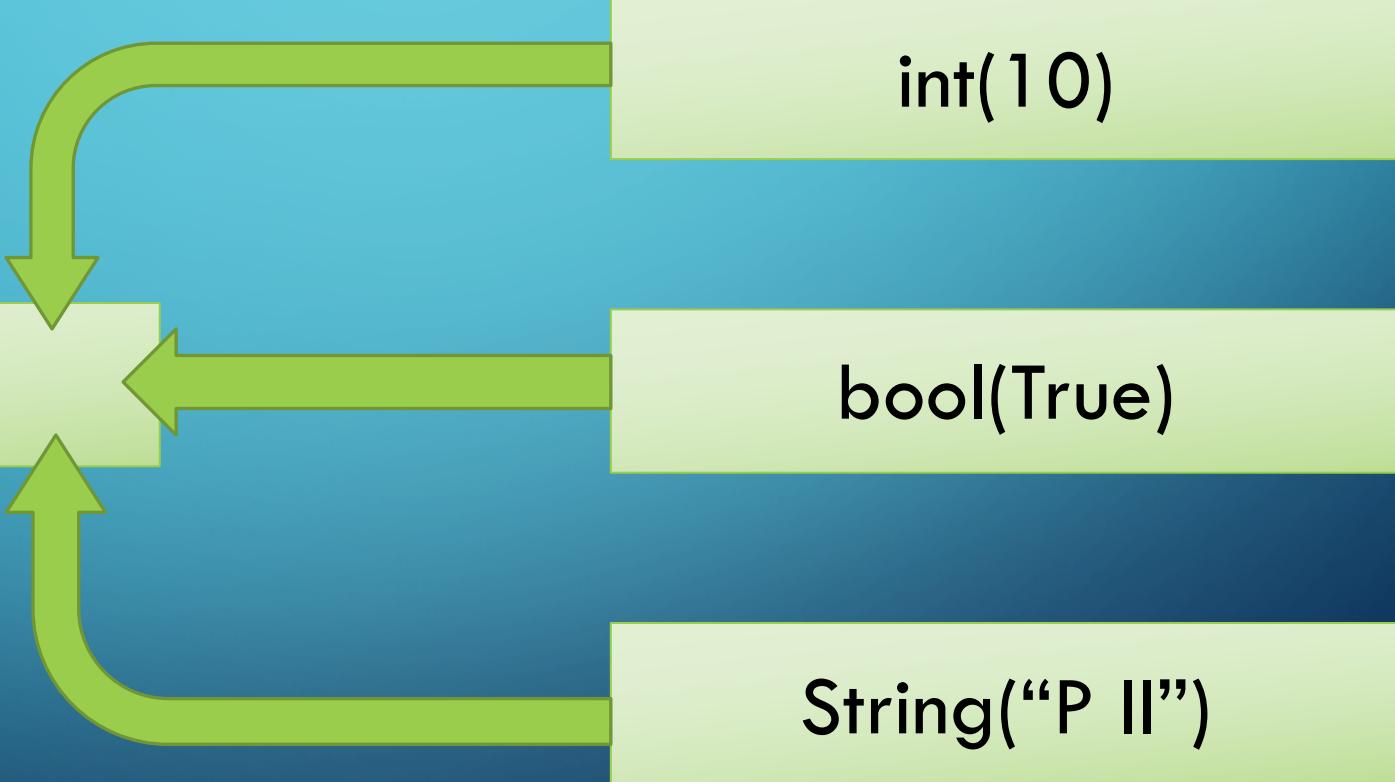
¿Que pasa en C?

int nombre\_var =



# BINDING DINÁMICO

nombre\_var =



# BINDING DINAMICO

nombre\_var =

0x7ffee5183a27



0x7ffee518286c

int(10)

bool(True)

0x7ffee6ad230f

String("P II")

0x7ffee5183a27

# BINDING DINAMICO

nombre\_var =

0x7ffee518286c

int(10)

0x7ffee518286c

bool(True)

0x7ffee6ad230f

String("P II")

0x7ffee5183a27



# REFERENCIAS – TIPOS SIMPLES

```
a = 10  
b = "P II"
```

```
a = b  
c = "P II"
```

```
d = 10  
e = 10 + 1  
f = "P II"
```

int(10)

0x7ffee518286c

bool(True)

0x7ffee6ad230f

String("P II")

0x7ffee5183a27

# REFERENCIAS – TIPOS SIMPLES

```
a = 10  
b = "P II"  
  
a = b  
c = "P II"  
  
d = 10  
e = 10 + 1  
f = "P II"
```

int(10)  
0x7ffee518286c

bool(True)  
0x7ffee6ad230f

String("P II")  
0x7ffee5183a27

# REFERENCIAS – TIPOS SIMPLES

a = 10 0x7ffee518286c

b = "P II" 0x7ffee5183a27

a = b 0x7ffee5183a27

c = "P II" 0x7ffee5183a27

d = 10 0x7ffee518286c

e = 10 + 1

f = "P II" 0x7ffee5183a27

int(10)

0x7ffee518286c

bool(True)

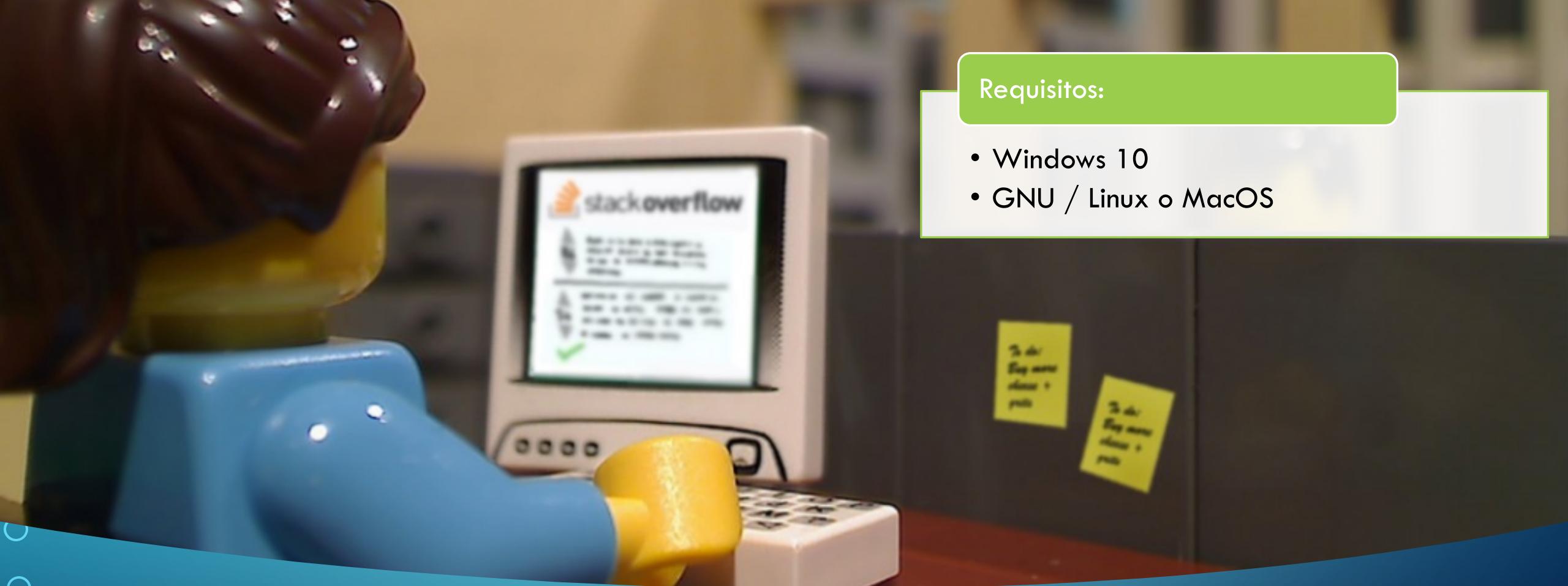
0x7ffee6ad230f

String("P II")

0x7ffee5183a27

## Requisitos:

- Windows 10
- GNU / Linux o MacOS



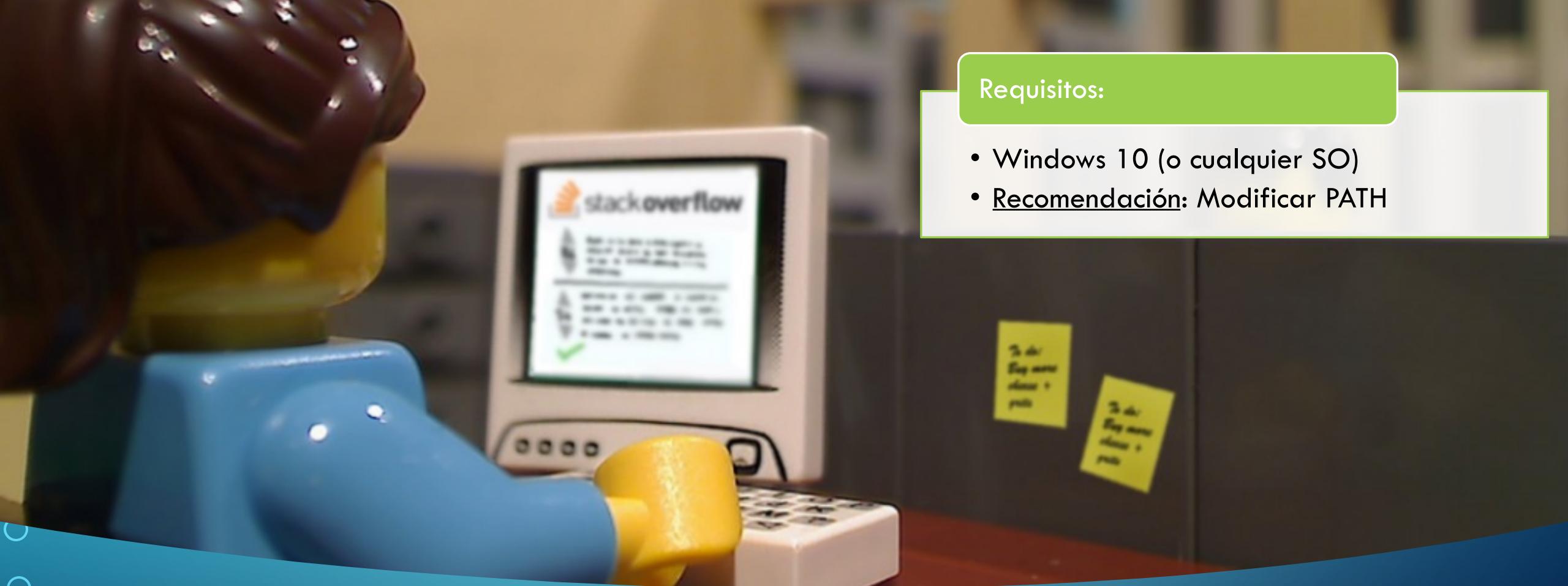
# PRACTICA PYTHON 3

- Creador Guido van Rossum a fines de los 80s
- En el 2000 se lanza la versión 2, que convivió mucho tiempo con la 3
- Actualmente se está migrando a la versión 3 estable
- Multiplataforma – Concepto de Maquina Virtual
- Virtualización: permite la aislación de diferentes ambientes de desarrollo.
- Diferentes IDEs (Integrated Development Environment)
  - PyCharm
  - Eclipse + Plugin Python – PyDev
- Tipado y dinámico
  - Corolario: todas las variables tienen un TIPO. Incluso “None” (a diferencia de Null en otros lenguajes, que hacen referencias a punteros). Función “type”

## INTRODUCCIÓN A PYTHON 3

## Requisitos:

- Windows 10 (o cualquier SO)
- Recomendación: Modificar PATH



# PYTHON 3 - INSTALACIÓN

# PYTHON 3 - DEMO

Implementación de Python 3.8+, IDE y VM para el sistema operativo Microsoft Windows 10.

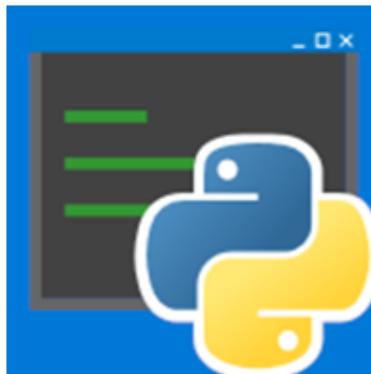
Hasta hace unos años, se mantenían las dos versiones. Actualmente, se desarrolla en 3, y a pedido del creador, no son backward compatible.

## Requisitos:

- Windows 10 (o cualquier SO)
- Recomendación: Modificar PATH

## Demo:

- Instalación pagina oficial
- Creación de virtualenv
- Instalación de paquetes en venv
- Trabajo en un IDE



# Python 3.8

Python Software Foundation • Herramientas de desarrollo > Kits de desarrollo

Compartir Lista de deseos

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal

Más

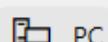
 TODOS

**Gratuito**

[Obtener](#) ...

[Información general](#)[Requisitos del sistema](#)[Opiniones](#)[Relacionado](#)

## Disponible en



PC

Welcome to Python.org

python.org

Python PSF Docs PyPI Jobs Community

# python™

Donate Search GO Socialize

About Downloads Documentation Community Success Stories News Events

```
# Python 3: Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!
```

```
# Input, assignment
>>> name = input('What is your name?\n')
>>> print('Hi, %s.' % name)
What is your name?
Python
Hi, Python.
```

Quick & Easy to Learn

Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. [Whet your appetite](#) with our Python 3 overview.

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. [» Learn More](#)

python-3.8.2.exe  
https://www.python.org  
0.2/25.3 MB, 23 mins left

Show all

Welcome to Python.org

python.org

Python PSF Docs PyPI Jobs Community

# python™

Donate Search GO Socialize

About Downloads Documentation Community Success Stories News Events

# Python 3: Lists  
  >>> fruits = ['  
  >>> loud\_fruits  
fruits]  
  >>> print(loud\_  
[ 'BANANA', 'APP  
  
# List and the  
  >>> list(enum  
[(0, 'Banana'),

All releases  
Source code  
Windows  
Mac OS X  
Other Platforms  
License  
Alternative Implementations

**Download for Windows**  
Python 3.8.2  
  
Note that Python 3.5+ cannot be used on Windows XP or earlier.  
Not the OS you are looking for? Python can be used on many operating systems and environments.  
View the full list of downloads.

Python is a programming language that lets you work quickly and integrate systems more effectively. [» Learn More](#)

<https://www.python.org/downloads/>

[Python](#)[PSF](#)[Docs](#)[PyPI](#)[Jobs](#)[Community](#)[Donate](#) Search[GO](#)[Socialize](#)[About](#)[Downloads](#)[Documentation](#)[Community](#)[Success Stories](#)[News](#)[Events](#)[Python](#) »» [Downloads](#) »» [Windows](#)

## Python Releases for Windows

- [Latest Python 3 Release - Python 3.8.2](#)
- [Latest Python 2 Release - Python 2.7.18](#)

### Stable Releases

- [Python 2.7.18 - April 20, 2020](#)
  - Download [Windows debug information files](#)
  - Download [Windows debug information files for 64-bit binaries](#)
  - Download [Windows help file](#)

### Pre-releases

- [Python 2.7.18rc1 - April 4, 2020](#)
  - Download [Windows debug information files](#)
  - Download [Windows debug information files for 64-bit binaries](#)
  - Download [Windows help file](#)
  - Download [Windows x86-64 MSI installer](#)

# Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
<a href="#">Gzipped source tarball</a>	Source release		f9f3768f757e34b342dbc06b41cbc844	24007411	<a href="#">SIG</a>
<a href="#">XZ compressed source tarball</a>	Source release		e9d6ebc92183a177b8e8a58cad5b8d67	17869888	<a href="#">SIG</a>
<a href="#">macOS 64-bit installer</a>	Mac OS X	for OS X 10.9 and later	f12203128b5c639dc08e5a43a2812cc7	30023420	<a href="#">SIG</a>
<a href="#">Windows help file</a>	Windows		7506675dcbb9a1569b54e600ae66c9fb	8507261	<a href="#">SIG</a>
<a href="#">Windows x86-64 embeddable zip file</a>	Windows	for AMD64/EM64T/x64	1a98565285491c0ea65450e78afe6f8d	8017771	<a href="#">SIG</a>
<a href="#">Windows x86-64 executable installer</a>	Windows	for AMD64/EM64T/x64	b5df1cbb2bc152cd70c3da9151cb510b	27586384	<a href="#">SIG</a>
<a href="#">Windows x86-64 web-based installer</a>	Windows	for AMD64/EM64T/x64	2586cdad1a363d1a8abb5fc102b2d418	1363760	<a href="#">SIG</a>
<a href="#">Windows x86 embeddable zip file</a>	Windows		1b1f0f0c5ee8601f160cfad5b560e3a7	7147713	<a href="#">SIG</a>
<a href="#">Windows x86 executable installer</a>	Windows		6f0ba59c7dbeba7bb0ee21682fe39748	26481424	<a href="#">SIG</a>
<a href="#">Windows x86 web-based installer</a>	Windows		04d97979534f4bd33752c183fc4ce680	1325416	<a href="#">SIG</a>

About

Downloads

Documentation

Community

Success Stories

News



## Install Python 3.8.2 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

### [Install Now](#)

C:\Users\nicap\AppData\Local\Programs\Python\Python38

Includes IDLE, pip and documentation  
Creates shortcuts and file associations

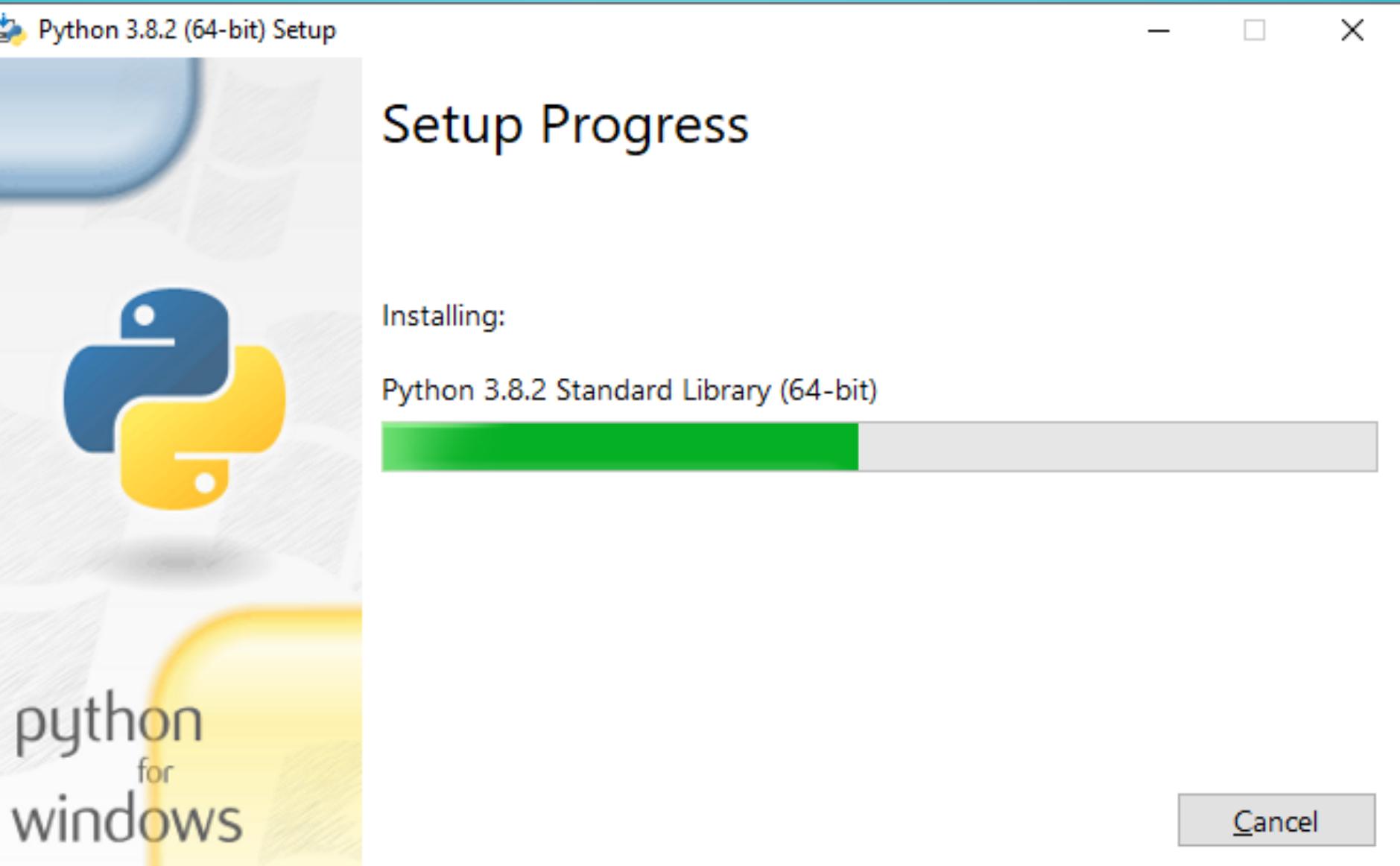
### [Customize installation](#)

Choose location and features

[Install launcher for all users \(recommended\)](#)

[Add Python 3.8 to PATH](#)

[Cancel](#)





## Ejecutar



Escriba el nombre del programa, carpeta, documento o recurso de Internet que desea abrir con Windows.

Abrir:

cmd



Aceptar

Cancelar

Examinar...

cmd. Seleccionar C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Versión 10.0.18363.778]  
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\nicap>python  
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> ^Z

C:\Users\nicap>

# RECOMENDACIÓN: AGREGAR AL PATH



# IDEs



Integrated Development Environment (en general)



IDLE: Integrated Development Environment (Python)



PyCharm – IntelliJ: Uno de los mas populares comercialmente



Eclipse + Python Plugin:  
Eclipse Foundation Open Source

Multiplataforma

Detección de errores

Configuraciones específicas



Gestión de Proyectos

Autocompletar

Reconocimiento de Sintaxis

Debugging

# TIPOS DE DATOS ESENCIALES

## Numericos

- **Integer:** Base 2, 8, 10, 16 (0b10011, 0o322, 12345, 0x8ab32)
- **Punto Flotante:** 255.0, 2.55e2, 2.55e-2
- **Numeros Complejos:** 2+4j

## Boolean

- True
- False

## Strings

- 'Cadena'
- ``Cadena''
- ``````Cadena``````

## Datetimes

- import datetime
- datetime.datetime.now()

# TIPOS DE DATOS ESENCIALES - NUMÉRICOS

## Enteros – int

- Base 2: prefijo 0b
- Base 8: prefijo 0o
- Base 16: prefijo 0x

## Reales, Punto Flotante – float

- Decimales: 0 . 34
- Notación Científica: 34e-2

## Complejos – complex

- Utilización del carácter " j "

```
>>> 12345  
12345  
>>> 0b100101  
37  
>>> 0o664  
436  
>>> 0xEBA432  
15442994  
  
>>> 23.434  
23.434  
>>> 5/4  
1.25  
>>> 5.56e-3  
0.00556  
  
>>> 5 + 7.5j  
(5+7.5j)
```

# TIPOS DE DATOS ESENCIALES - BOOLEAN

## Tipo bool

 Solo puede tomar dos valores

True

False

 Existe un mecanismo para evaluar otros tipos a bool (`__bool__`)



Permite combinarlos con otros tipos (lo vemos mas adelante...)

# TIPOS DE DATOS – OPCIONALES

## Fracciones

- Numerador
- Denominador

## Decimales

- Cuando se necesita exactitud
- Aplicaciones financieras y científicas

## Fecha y hora

```
>>> from fractions import Fraction  
>>> Fraction(45,25)  
Fraction(9, 5)  
  
>>> from decimal import Decimal  
>>> Decimal(34.4)  
Decimal('34.3999999999999857891452847979  
962825775146484375')  
  
>>> import datetime  
>>> datetime.datetime.now()  
datetime.datetime(2019, 7, 29, 12, 39, 6,  
241145)
```

- No existe el tipo “char” sino que es un string de longitud 1
  - “a” es un str y no un “char”
- Los strings son secuencias inmutables de code points Unicode 🤔
  - *Unicode: códigos numéricos de representación de caracteres (code points)*
  - *Asigna a cada carácter un código numérico*
  - [www.unicode.org](http://www.unicode.org)
- *El límite superior de Unicode es 0x10FFFF (1.114.111 caracteres posibles)*
- *Ejemplos:*
  - 😂 = U+1F475 = ‘CARA LLORANDO DE RISA’
  - N = U+004E = ‘LETRA N MAYUSCULA (EN ALFABETO LATINO)’

# TIPOS DE DATOS ESENCIALES - UNICODE

# TIPOS DE DATOS ESENCIALES - CODIFICACIÓN

- Pero el almacenamiento o transmisión de datos (textos) se produce en bytes. Por lo tanto es necesaria una codificación que establezca la relación entre code points y bytes.
- El valor (numérico) representado en esos bytes depende de la codificación utilizada (UTF, UCS)
- Python almacena los strings (secuencia de Unicode code points) como colección de bytes utilizando la codificación UTF-8. (8 bits para cada valor y más bytes si supera el límite)
- <https://docs.python.org/3/howto/unicode.html>



# TIPOS DE DATOS ESENCIALES – EJEMPLO UNICODE



# OPERADORES

Unarios



Aritméticos



Comparación



Lógicos



Bitwise



Asignación



Pertenencia



Identidad



Operando Izquierdo



Operando Derecho



Operador

# OPERADORES – UNARIOS

- `-` : Negación aritmética
- `+` : Identidad de Signo (**completitud**)

```
>>> -2342  
-2342  
>>> +4352  
4352
```

# OPERADORES - ARITMÉTICOS

- `+` : Suma aritmética
- `-` : Resta aritmética
- `*` : Multiplicación aritmética
- `/` : División aritmética (retorna punto flotante)
- `%` : Modulo (resto de la división)
- `**` : Exponenciación
- `//` : División Entera

```
>>> 234123 + 123213
```

```
357336
```

```
>>> 456 - 34533
```

```
-34077
```

```
>>> 234 * 12342134
```

```
2888059356
```

```
>>> 30 / 4
```

```
7.5
```

```
>>> 30 % 4
```

```
2
```

```
>>> 2 ** 20
```

```
1048576
```

```
>>> 30 // 4
```

```
7
```

# OPERADORES - COMPARACIÓN

- `==` : Igualdad de **Valores**
- `!=` : **Valores** no iguales
- `>` : **Mayor**
- `<` : **Menor**
- `<=` : **Mayor igual**
- `<=` : **Menor igual**

```
>>> 4 == 87
False
>>> 3 == 'hola'
False

>>> 4 != 87
True
>>> 3 != 'hola'
True

>>> 34 > 23
True
>>> 34 < 23
False
>>> 34 >= 23
True
>>> 34 <= 23
False
```

# OPERADORES - LÓGICOS

- and
  - Retorna True si ambas **Expresiones** evalúan como True
- or
  - Retorna True si alguna **Expresión** evalúan como True
- not
  - Niega la veracidad del valor

```
>>> (34 == 34) and ('h' > 'f')  
True
```

```
>>> (34 != 34) or False  
False
```

```
>>> not len([34,3,2]) == 5  
True
```

# OPERADORES - BITWISE

- & : **and**
- | : **or**
- ^ : **xor**
- ~ : **complemento**
- << : **left shift**
- >> : **right shift (completa con ceros)**

```
>>> bin(0b1100 & 0b0110)  
'0b100'
```

```
>>> bin(0b1100 | 0b0110)  
'0b1110'
```

```
>>> bin(0b1100 ^ 0b0110)  
'0b1010'
```

```
>>> bin(~0b1100)  
'-0b1101'  
>>> bin(~0b0110)  
'-0b111'
```

```
>>> bin(0b00110011 << 2)  
'0b11001100'
```

```
>>> bin(0b00110011 >> 2)  
'0b1100'
```

# OPERADORES - ASIGNACIÓN

- `=.` : Asignación simple a una variable
- `+=` : Aplica la suma y asigna el resultado
- `-=` : Aplica la resta y asigna el resultado
- `*=` : Aplica la multiplicación y asigna el resultado
- `/=` : Aplica la división y asigna el resultado
- `%=` : Aplica el módulo y asigna el resultado
- `**=` : Aplica la potencia y asigna el resultado
- `//=` : Aplica la división entera y asigna el resultado

```
>>> a = 5  
  
>>> a += 2  
a = 7  
  
>>> a -= 2  
a = 3  
  
>>> a *= 2  
a = 10  
  
>>> a /= 2  
a = 2.5  
  
>>> a %= 2  
a = 1  
  
>>> a **= 2  
a = 25  
  
>>> a //=2  
a = 2
```

# OPERADORES – PERTENENCIA E IDENTIDAD

- `in` : Determina si el operador de la izquierda se encuentra incluido en el operador de la derecha
- `not in` : Si no se encuentra incluido
- `is` : evalúa True si ambos operados son **IDENTICOS** (apuntan al mismo objeto – referencian al mismo puntero).
- `is not` : Si no son idénticos

```
>>> 34 in [54, 67, 34]
```

```
True
```

```
>>> 'arad' in  
'Paradigmas'
```

```
True
```

```
>>> 'Arad' in  
'Paradigmas'
```

```
False
```

```
>>> a = 3
```

```
>>> b = 4
```

```
>>> a is b
```

```
False
```

```
>>> a = b
```

```
>>> b is b
```

```
True
```

Dos variables pueden ser iguales pero no idénticas



# OPERADORES – PRECEDENCIA

( ) los paréntesis alteran precedencia

\*\*

~ + - (unarios)

\* / % //

+ - (binarios)

<< >>

&

^ |

<= < > >=

== !=

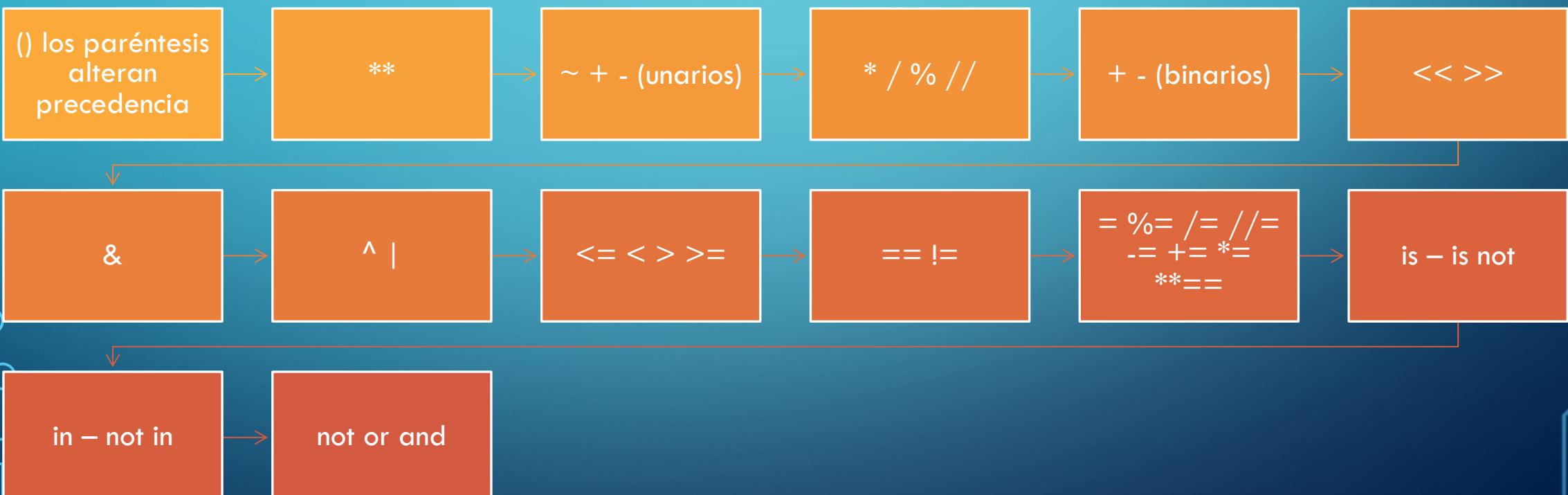
= %= /= //=. = += \*= \*\*==

is – is not

in – not in

not or and

# OPERADORES – PRECEDENCIA



# Colecciones - Inmutabilidad

## str (Strings)

- Inmutables.
- Permiten *templates*.

## tuple

- Inmutables
- Secuencia de objetos arbitrarios.

```
>>> materia = "Bienvenidos a {0}, somos {1} alumnos"  
>>> materia.format("Paradigmas II", "36")  
'Bienvenidos a Paradigmas II, somos 36 alumnos'  
  
>>> a,b,c = (12,"asd",23)  
a=12, b="asd", c = 23  
>>> a,b,c = c,a,b  
a = 23, b = 12, c = "asd"
```

# COLECCIONES

## list

- “**Mutables**”. Generalmente para elementos homogéneos.
- Operator overloading +, \*, -
- Sort, pop

## bytearrays

- Versión mutable de bytes.
- Elementos enteros [0, 256)

```
>>> a = list('Hola')
>>> a
['H', 'o', 'l', 'a']
>>> a.append('!')
>>> a
['H', 'o', 'l', 'a', '!']
>>> a.insert(3,'M')
>>> a
['H', 'o', 'l', 'M', 'a', '!']
>>> a.pop()
'!'
>>> a
['H', 'o', 'l', 'M', 'a']
```

# COLECCIONES SETS

**set (conjuntos)**  
**frozenset (inmutable)**

- Teoría de conjuntos
  - No duplicados
  - No importa el orden
- Necesidad de Hashability

```
>>> a = set()  
>>> a.add(12)  
>>> a.add('hola')  
>>> a.add(12.45)  
>>> a  
{'hola', 12, 12.45}  
  
>>> b = {12, 'que tal', 6/7} #otra notación  
>>> a.union(b)  
{0.8571428571428571, 'hola', 'que tal', 12,  
12.45}  
>>> a.intersection(b)  
{12}  
>>> a.difference(b)  
{'hola', 12.45}
```

# COLECCIONES DICTS

## dict (Diccionarios)

- Mapeo clave-valor (**key-values**)
- Las claves son únicas (**hashables**)
- Mutables

```
>>> a = {'a':1, 'b':'que tal',  
'c':45.56}  
>>> a.keys()  
dict_keys(['a', 'b', 'c'])  
>>> a.values()  
dict_values([1, 'que tal', 45.56])
```

# RANGOS

range: Muy útil para crear colecciones (rangos) indicando:

- El valor de inicio (inclusive)
- El valor final (exclusivo)
- El incremento (step)

```
>>> a = range(1,100,2)
1,3,5,...,99

>>> b = list(a)
>>> b
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, ..., 99]
>>> b[5:10]
[11, 13, 15, 17, 19]
>>> b[-1] # en C como seria?
99
```

# SLICING

**Slicing:** aplicable en todas las colecciones y secuencia

- Permite seleccionar un subconjunto o recorrerlo de otra manera
- Muy versátil, opciones de inicio, fin, incremento
- Crea una nueva colección

```
>>> c = "Hola que tal"  
>>> c[3:10]  
'a que t'  
>>> c[4:]  
' que tal'  
>>> c[:10]  
'Hola que t'  
>>> c[:] # es una nueva copia del string!  
'Hola que tal'  
>>> c[::-1]  
'lat euq aloH'
```

# ESTRUCTURAS – ESTÁTICAS Y DINÁMICAS

Estructura de datos

Forma de guardar y organizar la información, de forma eficiente y simple

Estáticas – Inmutables

No pueden modificarse una vez creadas

Tuples

Numbers

Strings

...

Dinámicas

Permiten modificaciones

List

Dict

Set

...

# ESTRUCTURAS DE CONTROL – IF/ELIF/ELSE



Branching:  
**if – elif – else**



## INDENTACION

- importante en Python
- 4 ESPACIOS

```
ingresos = 45500
coeficiente = 0.05

if ingresos < 30000:
    coeficiente = 0.1
elif ingresos < 66000:
    coeficiente = 0.15
else:
    coeficiente = 0.35

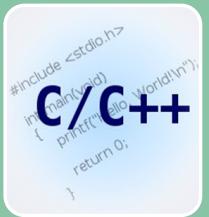
print("Resultado impuestos:", ingresos * coeficiente)
```

Resultado impuestos: 6825.0

# ESTRUCTURAS DE CONTROL – OPERADOR TERNARIO



Branching: if – else

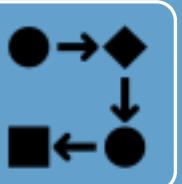


Único operador ternario  
similar al ?: de C

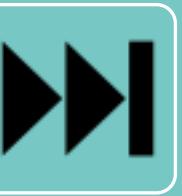
```
ingresos = 45500  
  
coeficiente = 0.1 if ingresos < 30000 else 0.15  
  
print("Resultado impuestos:", ingresos * coeficiente)  
  
# Otra notación  
coeficiente = (0.15, 0.1) [ingresos < 30000]  
  
print("Resultado impuestos:", ingresos * coeficiente)
```

Resultado impuestos: 6825.0

# ESTRUCTURAS DE ITERACIÓN – FOR



Looping: for



Itera sobre una secuencia hasta que se agota.



La forma de iteración se depende principalmente en la secuencia a iterar

```
for i in range(10):  
    print(i)
```

```
for i in range(1, 100, 2)[::-1]:  
    print(i)
```

```
99  
97  
95  
...  
5  
3  
1
```



# ESTRUCTURA FOR – EJEMPLOS

```
ciudades = ['Buenos Aires', 'Río de Janeiro', 'Denver']

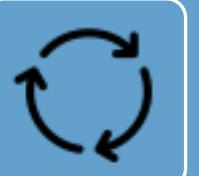
for ciudad in ciudades:
    print(ciudad)

for posicion, ciudad in enumerate(ciudades):
    print(posicion, ' - ' + ciudad)

paises = {
    'AR': 'Argentina',
    'BR': 'Brasil',
    'EU': 'Estados Unidos'
}
for pais in paises:
    print(pais)

for codigo, nombre in paises.items():
    print(codigo, ' -> ', nombre)
```

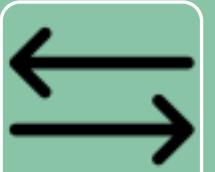
# ESTRUCTURAS DE ITERACIÓN – WHILE



Looping: while



Itera el bloque mientras sea verdadera (True) la condición



El bloque debería cambiar la condición a False para evitar entrar en un ciclo infinito

```
n = 57845
restos = []
while n > 0:
    remainder = n % 2
    restos.append(remainder)
    n //= 2

print(restos)
```



```
[1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1]
```

# ALTERACION EN EL FLUJO

A veces es necesario alterar el flujo normal de iteración, para poder omitir elementos (`continue`) o finalizar la iteración (`break`).

```
pares = []

for numero in range(1000):
    if numero % 2 != 0:
        continue
    else:
        pares.append(numero)

print(pares)
```



```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24,
26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46,
..., 994, 996, 998]
```

# ALTERACION EN EL FLUJO

A veces es necesario alterar el flujo normal de iteración, para poder omitir elementos (continue) o finalizar la iteración (break).

```
pares = []  
  
for numero in range(1000):  
    if numero % 2 != 0:  
        continue  
    else:  
        pares.append(numero)  
  
print(pares)
```

```
pares = list(filter(lambda x: x % 2 == 0, range(1000)))  
print(pares)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22,  
24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44,  
46, ..., 994, 996, 998]
```

# ESTRUCTURAS DE ITERACIÓN – ELSE

Particularidad de Python: La cláusula `else` se ejecuta si la iteración finaliza una vez agotada la secuencia (`for`) o finalizado el `while` (`condición false`)

```
paises = {  
    'AR': 'Argentina',  
    'BR': 'Brasil',  
    'EU': 'Estados Unidos'  
}  
  
for codigo, nombre in paises.items():  
    print(codigo, ' -> ', nombre)  
else:  
    print("Fin de la iteración, SIN BREAK")
```



```
AR -> Argentina  
BR -> Brasil  
EU -> Estados Unidos  
Fin de la iteración, SIN BREAK
```