



UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE INGENIERÍA INDUSTRIAL Y DE SISTEMAS

Programación para Machine Learning

INTRODUCCIÓN

NESTOR AUDANTE RAMOS

AGENDA

1. INTRODUCCIÓN
2. ESTRUCTURAS DE CONTROL
3. CADENAS
4. FUNCIONES
5. TUPLAS, LISTAS, ALIASING, MUTABILIDAD Y CLONES
6. RECURSIVIDAD Y DICCIONARIOS
7. EXCEPCIONES



AGENDA

1. **INTRODUCCIÓN**
2. ESTRUCTURAS DE CONTROL
3. CADENAS
4. FUNCIONES
5. TUPLAS, LISTAS, ALIASING, MUTABILIDAD Y CLONES
6. RECURSIVIDAD Y DICCIONARIOS
7. EXCEPCIONES





Lenguajes de Programación

→ Elementos primitivos

Números, cadenas de caracteres, operadores.



- ¿Que es la programación?

entrada

procedimiento

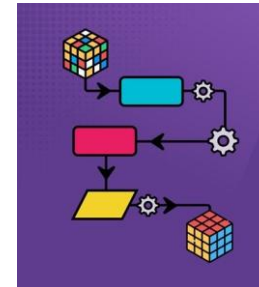
Salida

- Como una receta:

Consiste en instrucciones las cuales juntas producen un resultado

- Algoritmo:

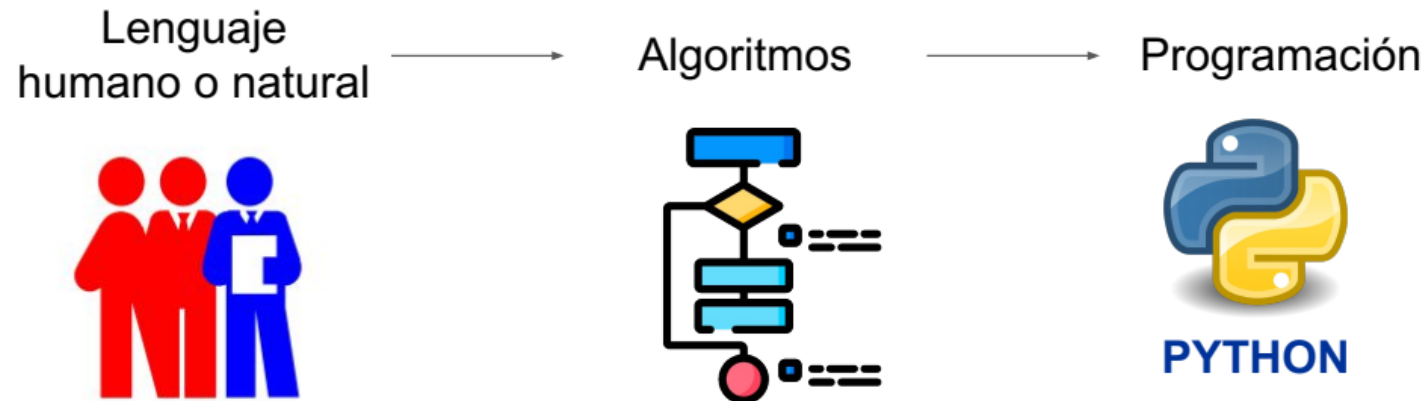
un conjunto de instrucciones que resuelven un problema



Objetivo de la disciplina



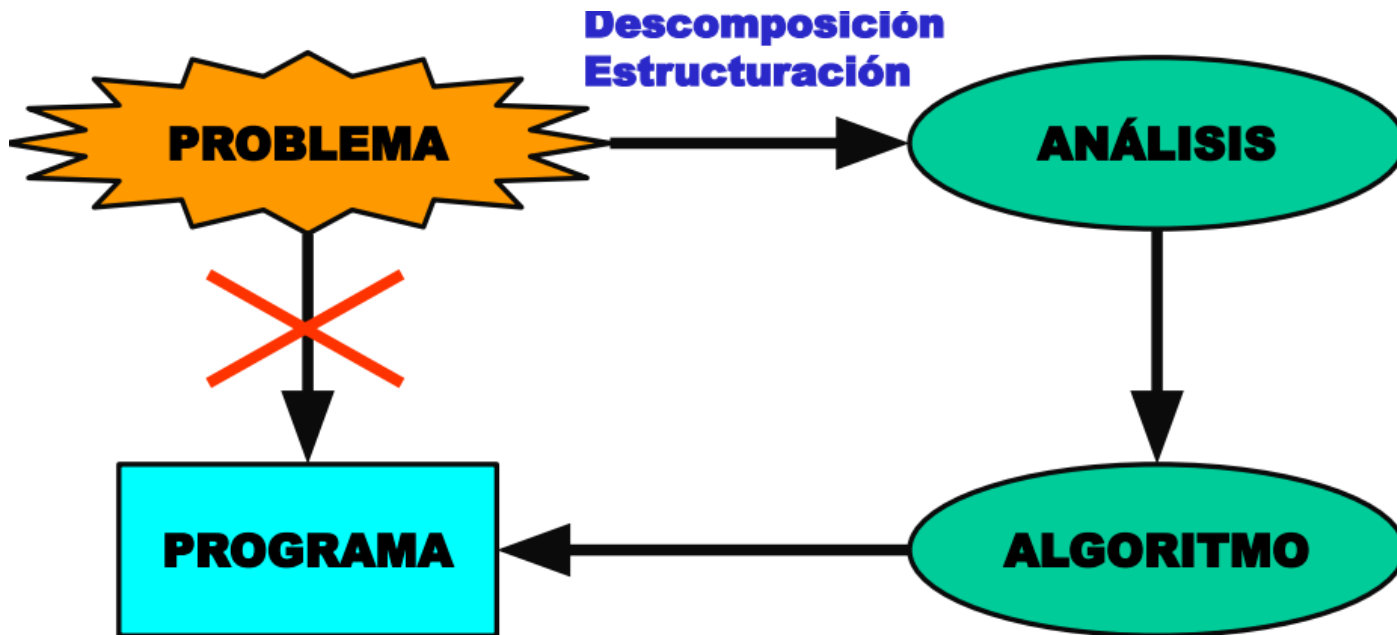
- Solución de problemas usando la computadora





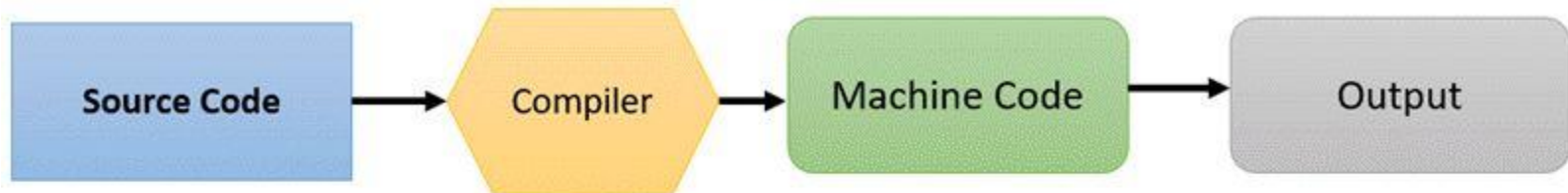
- Recopilación, análisis y especificación de requisitos.
- Algoritmo.
- Programa.
- Pruebas.
- Mantenimiento.

Ingeniería de Software





How Compiler Works



© guru99.com

How Interpreter Works

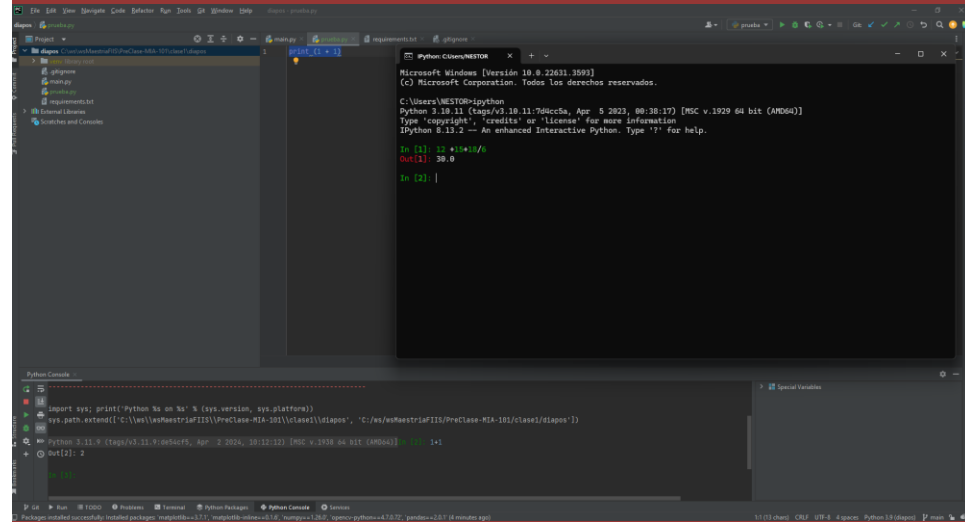




COMPARISON	COMPILER	INTERPRETER
Input	It takes an entire program at a time.	It takes a single line of code or instruction at a time.
Output	It generates intermediate object code.	It does not produce any intermediate object code.
Working mechanism	The compilation is done before execution.	Compilation and execution take place simultaneously.
Speed	Comparatively faster	Slower
Memory	Memory requirement is more due to the creation of object code.	It requires less memory as it does not create intermediate object code.
Errors	Display all errors after compilation, all at the same time.	Displays error of each line one by one.
Error detection	Difficult	Easier comparatively
Pertaining Programming languages	C, C++, C#, Scala, typescript uses compiler.	PHP, Perl, Python, Ruby uses an interpreter.

Programando en Python

- Un programa es una secuencia de **definiciones e instrucciones**.
- Las instrucciones inician una acción en el intérprete de Python.
- Los programas pueden ingresarse directamente en el **intérprete de comandos**, o a través de un archivo de texto.



The screenshot shows a Python IDE interface. On the left, a project named 'diapos' is open, containing files like 'diapos.py', 'diapos.ipynb', 'requirements.txt', and 'diapos.ipynb'. The main editor window displays a Python script with the following content:

```
import sys; print('Python ha on na' % (sys.version, sys.platform))
sys.path.append('C:\\Users\\nestor\\AppData\\Local\\Microsoft\\Windows\\Common-File\\diapos')
Python 3.11.9 (tags/v3.11.9:debf9, Apr 2 2024, 10:12:12) [MSC v.1918 64 bit (AMD64)]
> >>> 1+1
Out[1]: 2
```

The terminal window at the bottom shows the output of the script, including the Python version and platform information, and the result of the calculation 1+1, which is 2.



- Los programas en Python manipulan objetos.
- Cada objeto tiene asignado un tipo. El tipo define como un objeto interactúa con nuestro programa.
 - ◆ Pepe es un objeto del tipo estudiante. Pepe entonces puede `rendir_examen`, `asistir_a_clase` y `ir_al_comedor`.
 - ◆ Carlos es un objeto del tipo profesor. Carlos entonces puede `tomar_examen`, `reprobar_estudiante` y `dar_clase`.

Tipos de Objetos

- **int** para representar números enteros, como 5.
- **float** para representar números reales, como 3.27.
- **bool** para representar valores lógicos, como True o False.
- **NoneType** solo tiene un valor: None.
- Pueden usar `type()` para consultar el tipo de un objeto.



```
Console 1/A x
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC
v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more
information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [13]: type(5)
Out[13]: int

In [14]: type(3.27)
Out[14]: float

In [15]:

IPython console History
```

Conversión de Tipos



```
Console 1/A x
```

```
In [18]: type(3)
Out[18]: int

In [19]: float(3)
Out[19]: 3.0

In [20]: type(3.9)
Out[20]: float

In [21]: int(3.9)
Out[21]: 3

In [22]:
```

IPython console History



No se pueden utilizar **palabras reservadas** como nombres / identificadores de variables

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield



Expresiones en Python

- Se forman al combinar **objetos** y **operadores**.
- Tienen asignado un **valor**. Este valor tiene un **tipo**.
- Sintaxis:

`<objeto> <operador> <objeto>`



Sean dos objetos i y j

Sintaxis	Descripción	Valor
$i + j$	Adición	Si ambos son <code>int</code> , en resultado es <code>int</code> . Si al menos uno es <code>float</code> , el resultado es <code>float</code> .
$i - j$	Sustracción	
$i * j$	Multipliación	
i / j	División	El resultado es <code>float</code> .
$i \% j$	El residuo de dividir i entre j	
$i ** j$	El resultado de elevar i a la potencia j	

Nota: '//', división entera



Orden de Evaluación

1. Mediante **paréntesis**, se le puede indicar a Python qué evaluar primero.
2. Sin paréntesis:
 1. Exponenciación ($**$)
 2. Multiplicación ($*$)
 3. Adición ($+$) y sustracción ($-$) de izquierda a derecha.

Valores y Variables

- El signo igual (=) **asigna** un valor a una variable.
- Los valores están almacenados en memoria.
- Para obtener el valor asociado a una variable, basta escribir su nombre.
- Facilita reutilizar valores, y cambiarlos fácilmente en caso sea necesario.



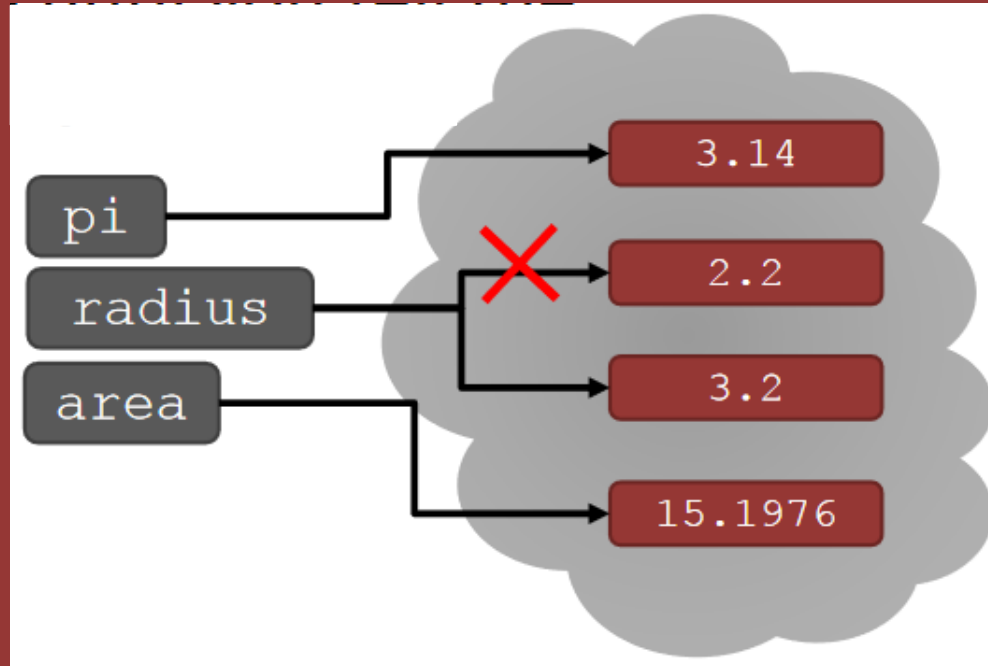
```
Console 1/A x
In [29]: pi = 3.14159
In [30]: radio = 2.2
In [31]: area = pi * (radio ** 2)
In [32]: print(area)
15.205295600000001
In [33]:
```

IPython console History

Re-assignar Variables

```
Console 1/A x
In [35]: pi = 3.14
In [36]: radius = 2.2
In [37]: area = pi * (radius ** 2)
In [38]: radius = radius + 1
In [39]: print(radius)
3.2
In [40]:
```

IPython console History



Cadenas de Caracteres

- Incluyen letras, caracteres especiales, espacios en blanco y dígitos.
- Definidas entre `'comillas simples'` o `"comillas dobles"`
- Para concatenar, utilizar `+`.
- Para mostrar texto en consola, utilizar `print()`.

```
1 hola = "Hola"
2 nombre = "Francisco"
3
4 saludo = hola + nombre
5 print(saludo)
6
7 saludo = hola + " " + nombre
8 print(saludo)
9
10 saludito = (hola + " ") * 3 + nombre
11 print(saludito)
```

IPython console

Console 1/A

```
In [11]: runfile('D:/git-bic01/primer-
laboratorio-los_jotitas/untitled1.py',
wdir='D:/git-bic01/primer-laboratorio-
los_jotitas')
HolaFrancisco
Hola Francisco
Hola Hola Hola Francisco
```

```
In [12]: |
```

Input()

- Primero, muestra en consola el texto entre comillas.
- El usuario luego puede ingresar un valor.
- Luego de presionar `Enter`, el valor queda asociado a una variable.
- El valor almacenado de una cadena de caracteres.

```
1 numero_como_cadena = input("Ingrese un numero: ")
2 numero_como_int = int(numero_como_cadena)
3
4 resultado = numero_como_int * 2
5 print("El doble de " + numero_como_cadena + " es "
6       + str(resultado))
```

IPython console

Console 1/A

```
In [14]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled1.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
```

```
Ingrese un numero: 10
El doble de 10 es 20
```

```
In [15]: |
```

AGENDA

1. INTRODUCCIÓN
2. **ESTRUCTURAS DE CONTROL**
3. CADENAS
4. FUNCIONES
5. TUPLAS, LISTAS, ALIASING, MUTABILIDAD Y CLONES
6. RECURSIVIDAD Y DICCIONARIOS
7. EXCEPCIONES
8. PROGRAMACION ORIENTADA A OBJETOS





Producen valores booleanos. Sean dos variables i y j ,

Sintaxis	Descripción
$i > j$	True si i es mayor que j . False caso contrario.
$i \geq j$	True si i es mayor o igual que j . False caso contrario.
$i < j$	True si i es menor que j . False caso contrario.
$i \leq j$	True si i es menor o igual que j . False caso contrario.
$i == j$	True si i es igual a j . False caso contrario.
$i \neq j$	True si i es diferente de j . False caso contrario.



Sean dos variables a y b que contienen valores booleanos:

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Un Ejemplo



```
1  numero_cervezas = 3
2  minimo_legal = 1
3
4  puedo_manejar = numero_cervezas < minimo_legal
5  print(puedo_manejar)
6
7  efectivo = 15
8  costo_taxi = 10
9
10 tomar_taxi = not puedo_manejar and efectivo >= costo_taxi
11 print(tomar_taxi)
```

IPython console

Console 1/A

```
In [20]: runfile('D:/git-bic01/primer-laboratorio-los_jotitas/
untitled1.py', wdir='D:/git-bic01/primer-laboratorio-los_jotitas')
False
True

In [21]: |
```

Sentencias Condicionales

```
if <condition>:  
    <expression>  
    <expression>  
    ...
```

```
if <condition>:  
    <expression>  
    <expression>  
    ...  
else:  
    <expression>  
    <expression>  
    ...
```

```
if <condition>:  
    <expression>  
    <expression>  
    ...  
elif <condition>:  
    <expression>  
    <expression>  
    ...  
else:  
    <expression>  
    <expression>  
    ...
```

Indentación

En Python, usamos indentación para definir bloques de código.

```
1 x = float(input("Ingrese el valor de x: "))
2 y = float(input("Ingrese el valor de y: "))
3
4 if x == y:
5     print("Ambos numeros son iguales")
6
7     if y != 0:
8         print("El valor de x/y es ", x/y)
9 elif x < y:
10    print("x es menor que y")
11 else:
12    print("y es menor que x")
13
14 print("F I N")
15
```

IPython console

Console 1/A

```
In [23]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled1.py', wdir='D:/git-bic01/primer-
laboratorio-los_jotitas')
```

Ingrese el valor de x: 4

Ingrese el valor de y: 4

Ambos numeros son iguales

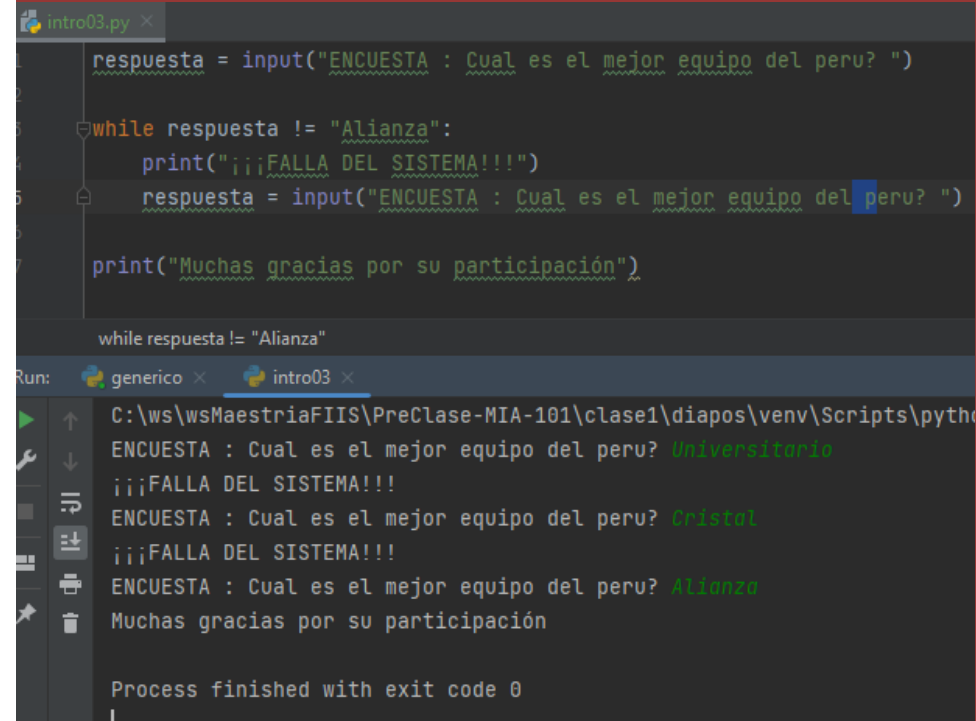
El valor de x/y es 1.0

F I N

In [24]:

Iteración con while

1. La condición después de `while` produce un valor booleano.
2. Si el valor es `True`, se ejecutan las instrucciones dentro del bloque de código.
3. Se evalúa la condición nuevamente.
4. El proceso continúa hasta que la condición produce un valor `False`.



The screenshot displays a Python IDE with a file named `intro03.py`. The code defines a `while` loop that prompts the user for the best Peruvian football team. If the answer is not "Alianza", it prints a system failure message and asks again. Once "Alianza" is entered, it prints a thank-you message and ends the loop.

```
1 respuesta = input("ENCUESTA : Cual es el mejor equipo del peru? ")
2
3 while respuesta != "Alianza":
4     print("!!!FALLA DEL SISTEMA!!!")
5     respuesta = input("ENCUESTA : Cual es el mejor equipo del peru? ")
6
7 print("Muchas gracias por su participación")
```

The console output shows the program's execution with user inputs in green:

```
Run: generico x intro03 x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe
ENCUESTA : Cual es el mejor equipo del peru? Universitario
!!!FALLA DEL SISTEMA!!!
ENCUESTA : Cual es el mejor equipo del peru? Cristal
!!!FALLA DEL SISTEMA!!!
ENCUESTA : Cual es el mejor equipo del peru? Alianza
Muchas gracias por su participación

Process finished with exit code 0
```

Iteración con for

- En cada iteración, se le asigna un valor a la variable `contador`.
- La primera iteración, a `contador` se le asigna el valor de 0.
- La siguiente iteración, el valor de `contador` se incrementa en 1.
- Continuar hasta que `contador == 5 - 1`.

```
1 print("Iteracion con while:")
2 contador = 0
3 while contador < 5:
4     print(contador)
5     contador = contador + 1
6
7 print("Iteracion con for:")
8 for contador in range(5):
9     print(contador)
10
```

IPython console

Console 1/A

```
In [36]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled1.py', wdir='D:/git-bic01/primer-
laboratorio-los_jotitas')
Iteracion con while:
0
1
2
3
4
Iteracion con for:
0
1
2
3
4

In [37]:
```

`range(inicio, fin, incremento)`

- `inicio` e `incremento` son opcionales. Sus valores por defecto son `inicio = 0` e `incremento = 1`.
- La iteración continúa hasta llegar al valor `fin - 1`.

```
1 total = 0
2
3 for numero in range(5, 11, 2):
4     print("numero: ", numero)
5     total += numero
6
7 print("total ", total)
```

IPython console

Console 1/A

```
In [38]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled1.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
```

```
numero: 5
numero: 7
numero: 9
total 21
```

```
In [39]: |
```

break

- Sale inmediatamente del bucle en el que se encuentre.
- Cualquier expresión posterior **no** es ejecutada.
- `break` solo termina el **bucle interno** que la contiene.

```
1 total = 0
2
3 for numero in range(5, 11, 2):
4     print("numero: ", numero)
5     total += numero
6
7     if total == 5:
8         break
9     total += 1
10
11 print("total ", total)
```

IPython console

Console 1/A

```
In [41]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled1.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
numero: 5
total 5
```

```
In [42]: |
```


while y for

for

- Número de iteraciones conocido.
- Puede finalizarse utilizando `break`.
- Incluye un contador.
- Es posible implementar la misma lógica con `while`.

while

- Número de iteraciones sin restricciones.
- Puede finalizarse utilizando `break`.
- Un contador debe implementarse manualmente.
- No siempre pueden implementarse mediante `for`.

AGENDA

1. INTRODUCCIÓN
2. ESTRUCTURAS DE CONTROL
3. **CADENAS**
4. FUNCIONES
5. TUPLAS, LISTAS, ALIASING, MUTABILIDAD Y CLONES
6. RECURSIVIDAD Y DICCIONARIOS
7. EXCEPCIONES
8. PROGRAMACION ORIENTADA A OBJETOS



Cadenas de Caracteres

- `len()` obtiene el número de caracteres de la cadena entre paréntesis.
- Mediante corchetes `[]`, podemos obtener caracteres de acuerdo a su posición (o índice).
- De izquierda a derecha, el índice empieza en 0.
- De derecha a izquierda, el índice empieza en -1.

```
1 una_cadena = "UNI"
2
3 print("len(una_cadena):", len(una_cadena))
4
5 print("una_cadena[0]:", una_cadena[0])
6 print("una_cadena[1]:", una_cadena[1])
7 print("una_cadena[2]:", una_cadena[2])
8 print("una_cadena[-1]:", una_cadena[-1])
9 print("una_cadena[-2]:", una_cadena[-2])
10 print("una_cadena[-3]:", una_cadena[-3])
11
12 print("una_cadena[3]:", una_cadena[3])
13
```

IPython console

Console 1/A

```
In [6]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
len(una_cadena): 3
una_cadena[0]: U
una_cadena[1]: N
una_cadena[2]: I
una_cadena[-1]: I
una_cadena[-2]: N
una_cadena[-3]: U
Traceback (most recent call last):

  File "D:\git-bic01\primer-laboratorio-los_jotitas
\untitled0.py", line 12, in <module>
    print("una_cadena[3]:", una_cadena[3])

IndexError: string index out of range
```

Slicing

- Podemos obtener partes de la cadena mediante [inicio: fin: incremento].
- Si solo indicamos [inicio: fin], incremento toma el valor de 1.
- De derecha a izquierda, el índice empieza en -1.
- Es posible omitir valores.

```
1 una_cadena = "abcdefgh"
2
3 print("una_cadena[3:6]: " + una_cadena[3:6])
4 print("una_cadena[3:6:2]: " + una_cadena[3:6:2])
5 print("una_cadena[:]: " + una_cadena[:])
6 print("una_cadena[::-1]: " + una_cadena[::-1])
7 print("una_cadena[4:1:-2]: " + una_cadena[4:1:-2])
8
```

IPython console

Console 1/A

```
In [11]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/primer-
laboratorio-los_jotitas')
una_cadena[3:6]: def
una_cadena[3:6:2]: df
una_cadena[:]: abcdefgh
una_cadena[::-1]: hgfedcba
una_cadena[4:1:-2]: ec

In [12]: |
```

Las Cadenas son Inmutables

```
1 s = "hello"
2 print(s)
3
4 s = 'y' + s[1:len(s)]
5 print(s)
6
7 s[0] = 'z'
8
9
```

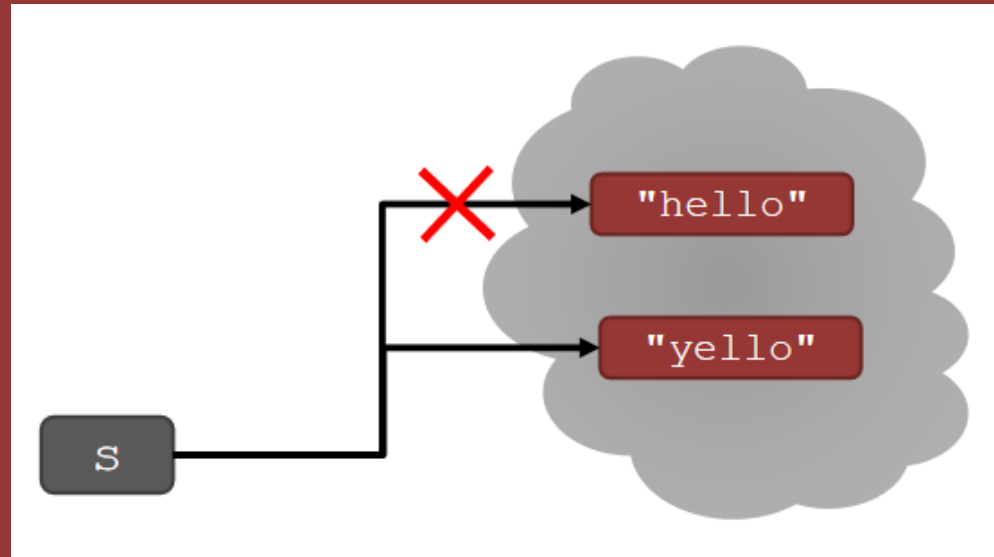
IPython console

Console 1/A

```
In [14]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
hello
yello
Traceback (most recent call last):

  File "D:\git-bic01\primer-laboratorio-los_jotitas
\untitled0.py", line 8, in <module>
    s[0] = 'z'

TypeError: 'str' object does not support item
assignment
```



Cadenas y Bucles

- Ambos bucles `for` realizan lo mismo.
- El bucle de la línea 7 se considera más idiomático.

```
1  una_cadena = "abcdefgh"
2
3  for indice in range(len(una_cadena)):
4      if una_cadena[indice] == "e":
5          print("La cadena contiene la letra 'e'")
6
7  for letra in una_cadena:
8      if letra == "e":
9          print("La cadena contiene la letra 'e'")
10
11
```

IPython console

Console 1/A

```
In [17]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/primer-
laboratorio-los_jotitas')
La cadena contiene la letra 'e'
La cadena contiene la letra 'e'

In [18]: |
```

AGENDA

1. INTRODUCCIÓN
2. ESTRUCTURAS DE CONTROL
3. CADENAS
4. **FUNCIONES**
5. TUPLAS, LISTAS, ALIASING, MUTABILIDAD Y CLONES
6. RECURSIVIDAD Y DICCIONARIOS
7. EXCEPCIONES





- Un programa puede ser descompuesto en módulos:
 - ◆ Son independientes.
 - ◆ Permiten dividir un programa en partes.
 - ◆ Son reutilizables.
 - ◆ Permiten organizar un programa.
- Podemos descomponer un programa mediante funciones y clases.

Funciones

- Porciones de código reutilizable.
- Para ser ejecutada, una función debe ser **invocada**.
- Elementos:
 - ◆ Nombre.
 - ◆ Parámetros (0 o más).
 - ◆ Docstring (opcional, pero recomendada).
 - ◆ Cuerpo de la función.
 - ◆ Dato de retorno.

```
1 def es_numero_par(numero):  
2     """  
3     Entrada: numero, un entero positivo.  
4     Salida: True si numero es par, False caso contrario.  
5  
6     """  
7  
8     print("Evaluando ", numero)  
9     return numero % 2 == 0  
10  
11 resultado = es_numero_par(3)  
12 print(resultado)
```

IPython console



Console 1/A ×

Console 2/A ×



```
In [3]: runfile('D:/git-bic01/primer-laboratorio-  
los_jotitas/untitled0.py', wdir='D:/git-bic01/primer-  
laboratorio-los_jotitas')  
Evaluando 3  
False  
  
In [4]: |
```

Ámbito

- Al momento de invocar la función con $f(x)$, el parámetro x toma el valor de 3.
- Cuando ejecutamos una función, creamos un nuevo **ámbito**.
- El valor de una variable **depende del ámbito** en el que se encuentre.

```
1 def f(x):  
2     x = x + 1  
3     print("En f(x). x=", x)  
4     return x  
5  
6 x = 3  
7 z = f(x)  
8  
9 print("x=", x)  
10 print("z=", z)  
11  
12
```

IPython console

Console 2/A

```
In [8]: runfile('D:/git-bic01/primer-laboratorio-  
los_jotitas/untitled0.py', wdir='D:/git-bic01/  
primer-laboratorio-los_jotitas')  
En f(x). x= 4  
x= 3  
z= 4  
  
In [9]: |
```

Iniciando Línea 7

```
1  def f(x):  
2      x = x + 1  
3      print("En f(x). x=", x)  
4      return x  
5  
6  x = 3  
7  z = f(x)  
8  
9  print("x=", x)  
10 print("z=", z)  
11  
12
```

Global scope

f

Some
code

x

3

z

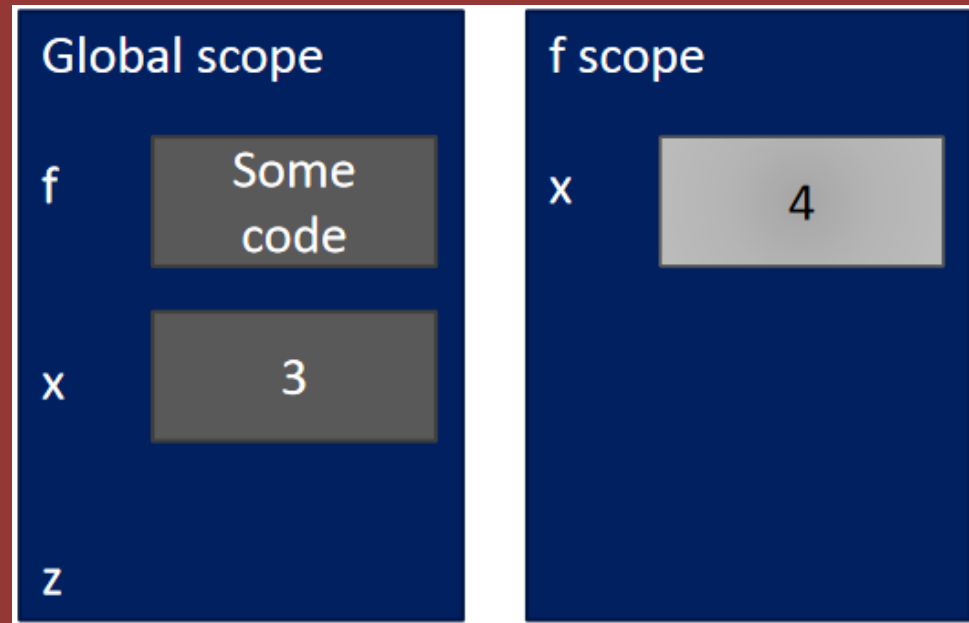
f scope

x

3

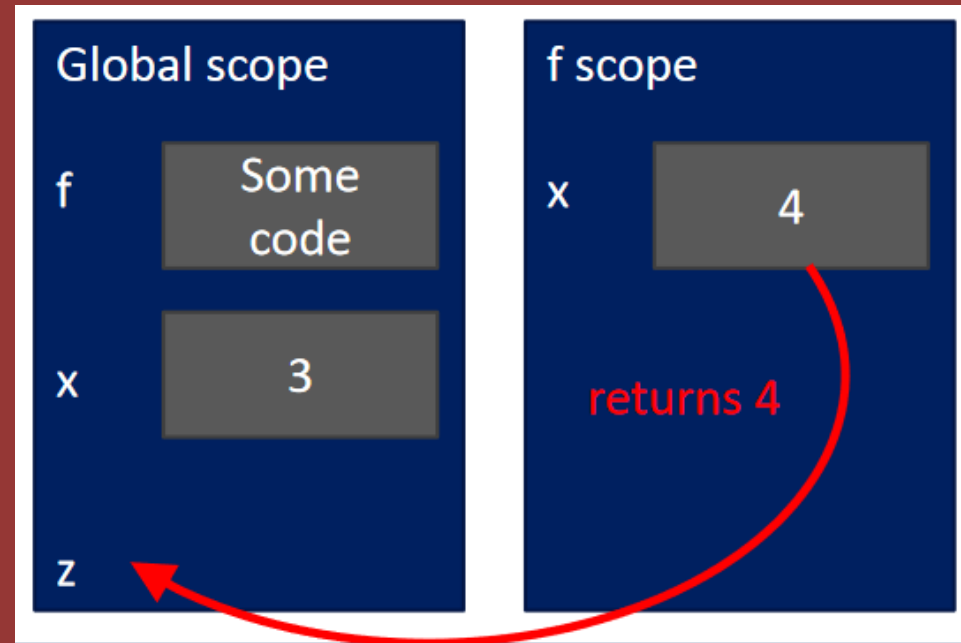
Ejecutando f(3). Línea 2

```
1  def f(x):  
2      x = x + 1  
3      print("En f(x). x=", x)  
4      return x  
5  
6  x = 3  
7  z = f(x)  
8  
9  print("x=", x)  
10 print("z=", z)  
11  
12
```



Ejecutando f(3). Línea 4

```
1  def f(x):  
2      x = x + 1  
3      print("En f(x). x=", x)  
4      return x  
5  
6  x = 3  
7  z = f(x)  
8  
9  print("x=", x)  
10 print("z=", z)  
11  
12
```



Línea 7 ejecutada

```
1  def f(x):  
2      x = x + 1  
3      print("En f(x). x=", x)  
4      return x  
5  
6  x = 3  
7  z = f(x)  
8  
9  print("x=", x)  
10 print("z=", z)  
11  
12
```

Global scope

f

Some
code

x

3

z

4

return

- En caso la función no utilice return, Python producirá None como salida.
- None representa la ausencia de un valor.

```
1 def es_numero_par(numero):  
2     """  
3     Entrada: numero, un entero positivo.  
4     Salida: True si numero es par, False caso  
5     """  
6  
7  
8     print("Evaluando ", numero)  
9     numero % 2 == 0  
10  
11 resultado = es_numero_par(3)  
12 print(resultado)
```

IPython console

Console 2/A

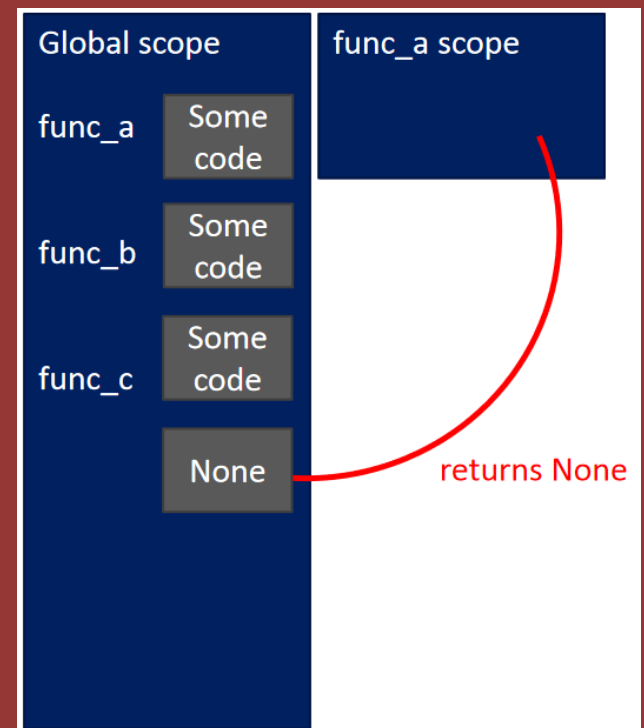
```
In [13]: runfile('D:/git-bic01/primer-laboratorio-  
los_jotitas/untitled0.py', wdir='D:/git-bic01/  
primer-laboratorio-los_jotitas')  
Evaluando 3  
None
```

```
In [14]: |
```

Línea 12 Ejecutada

```
README.md x untitled0.py* x
1  def func_a():
2      print("Ejecutando func_a")
3
4  def func_b(y):
5      print("Ejecutando func_b")
6      return y
7
8  def func_c(z):
9      print("Ejecutando func_c")
10     return z()
11
12  print(func_a())
13  print(5 + func_b(2))
14  print(func_c(func_a))
15

IPython console
Console 2/A x
In [14]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
Ejecutando func_a
None
```



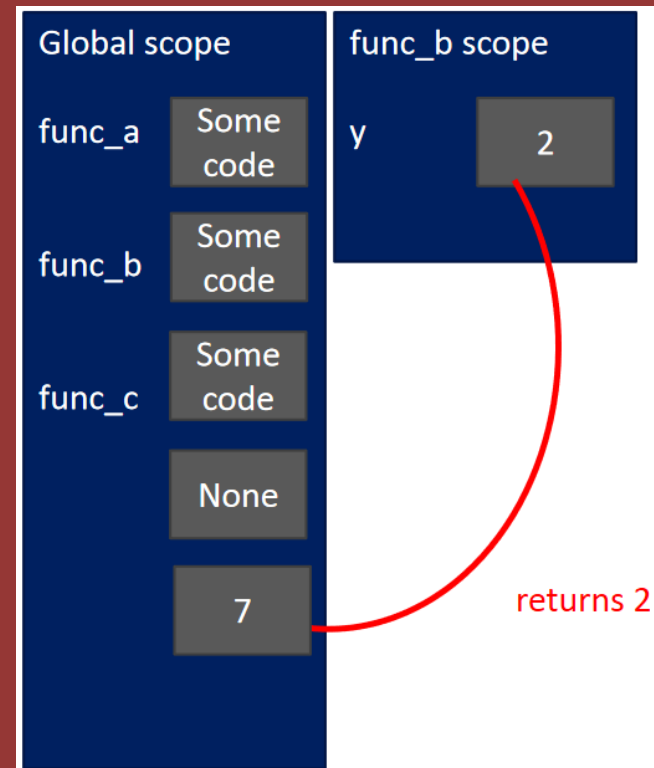
Línea 13 Ejecutada

```
1 def func_a():
2     print("Ejecutando func_a")
3
4 def func_b(y):
5     print("Ejecutando func_b")
6     return y
7
8 def func_c(z):
9     print("Ejecutando func_c")
10    return z()
11
12 print(func_a())
13 print(5 + func_b(2))
14 print(func_c(func_a))
15
```

IPython console

Console 2/A

```
In [14]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
Ejecutando func_a
None
Ejecutando func_b
7
```



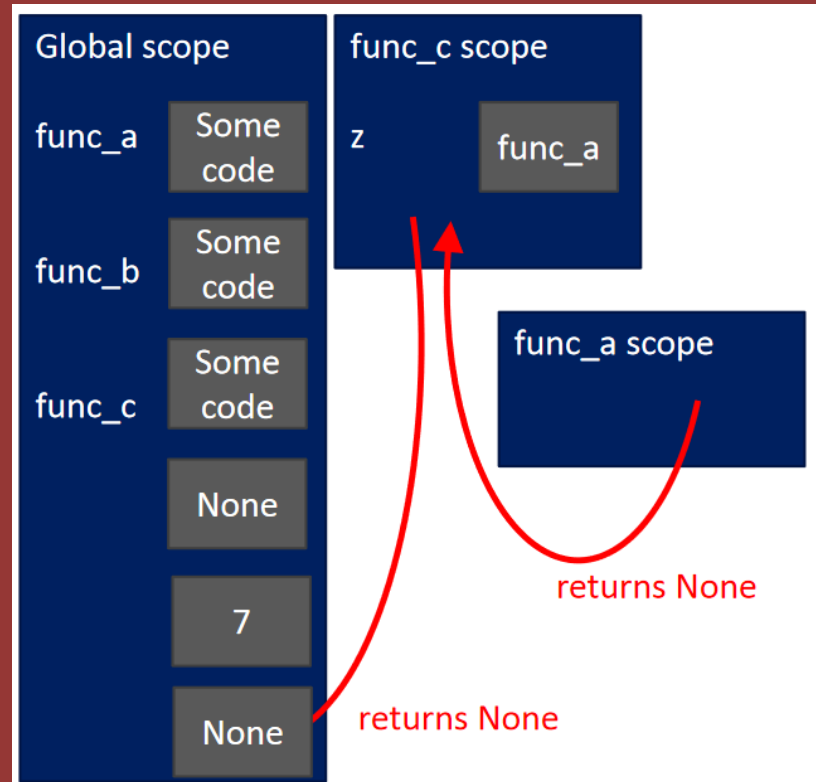
Funciones como Parámetros

```
1 def func_a():
2     print("Ejecutando func_a")
3
4 def func_b(y):
5     print("Ejecutando func_b")
6     return y
7
8 def func_c(z):
9     print("Ejecutando func_c")
10    return z()
11
12 print(func_a())
13 print(5 + func_b(2))
14 print(func_c(func_a))
15
```

IPython console

Console 2/A

```
In [14]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
Ejecutando func_a
None
Ejecutando func_b
7
Ejecutando func_c
Ejecutando func_a
None
```



Ámbito

- Una función tiene acceso a variables definidas externamente.
- Pero una función **no puede modificar** variables definidas externamente.

```
1 def una_funcion(parametro):
2     variable = 1
3     variable += 1
4     print("una_funcion: variable=", variable)
5
6 def otra_funcion(parametro):
7     print("otra_funcion: variable=", variable)
8     print("otra_funcion: variable+1=", variable + 1)
9
10 def la_funcion(parametro):
11     variable += 1
12
13
14 variable = 5
15
16 una_funcion(variable)
17 print("variable =", variable)
18 otra_funcion(variable)
19 print("variable =", variable)
20 la_funcion(variable)
```

IPython console

Console 2/A

```
una_funcion: variable= 2
variable = 5
otra_funcion: variable= 5
otra_funcion: variable+1= 6
variable = 5
Traceback (most recent call last):

  File "D:\git-bic01\primer-laboratorio-los_jotitas\untitled0.py", line 20, in <module>
    la_funcion(variable)

  File "D:\git-bic01\primer-laboratorio-los_jotitas
```

AGENDA

1. INTRODUCCIÓN
2. ESTRUCTURAS DE CONTROL
3. CADENAS
4. FUNCIONES
5. **TUPLAS, LISTAS, ALIASING, MUTABILIDAD Y CLONES**
6. RECURSIVIDAD Y DICCIONARIOS
7. EXCEPCIONES



Tuplas

- Son una secuencia ordenada de elementos, que pueden tener distintos tipos.
- Son **inmutables**, como las cadenas de caracteres. No se pueden alterar los valores de una tupla.
- Se representan mediante paréntesis ().

```
1  tupla_vacia = ()
2
3  una_tupla = (2, "UNI", 3)
4  print("una_tupla[0] ", una_tupla[0])
5  print("len(una_tupla) ", len(una_tupla))
6  print("una_tupla[1:2] ", una_tupla[1:2])
7  print("una_tupla[1:3] ", una_tupla[1:3])
8
9  otra_tupla = (2, "UNI", 3) + (5, 6)
10 print("otra_tupla ", otra_tupla)
11 otra_tupla[1] = 4
12
```

IPython console

Console 1/A

```
In [5]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/primer-
laboratorio-los_jotitas')
una_tupla[0] 2
len(una_tupla) 3
una_tupla[1:2] ('UNI',)
una_tupla[1:3] ('UNI', 3)
otra_tupla (2, 'UNI', 3, 5, 6)
Traceback (most recent call last):

  File "D:\git-bic01\primer-laboratorio-los_jotitas
\untitled0.py", line 12, in <module>
    otra_tupla[1] = 4

TypeError: 'tuple' object does not support item
assignment
```

Usando Tuplas

- Ofrecen una manera fácil de intercambiar valores entre variables.
- Nos permiten devolver más de un valor en una función.

```
1 una_variable = 10
2 otra_variable = 3
3
4 una_variable = otra_variable
5 otra_variable = una_variable
6 print("una_variable ", una_variable)
7 print("otra_variable ", otra_variable)
8
9 una_variable = 10
10 otra_variable = 3
11 (una_variable, otra_variable) = (otra_variable, una_variable)
12 print("una_variable ", una_variable)
13 print("otra_variable ", otra_variable)
14
15 def division_entera(dividendo, divisor):
16     cociente = dividendo // divisor
17     residuo = dividendo % divisor
18     return (cociente, residuo)
19
20 print(division_entera(10, 3))
```

Python console

Console 1/A

```
In [7]: runfile('D:/git-bic01/primer-laboratorio-los_jotitas/
untitled0.py', wdir='D:/git-bic01/primer-laboratorio-los_jotitas')
una_variable 3
otra_variable 3
una_variable 3
otra_variable 10
(3, 1)
```

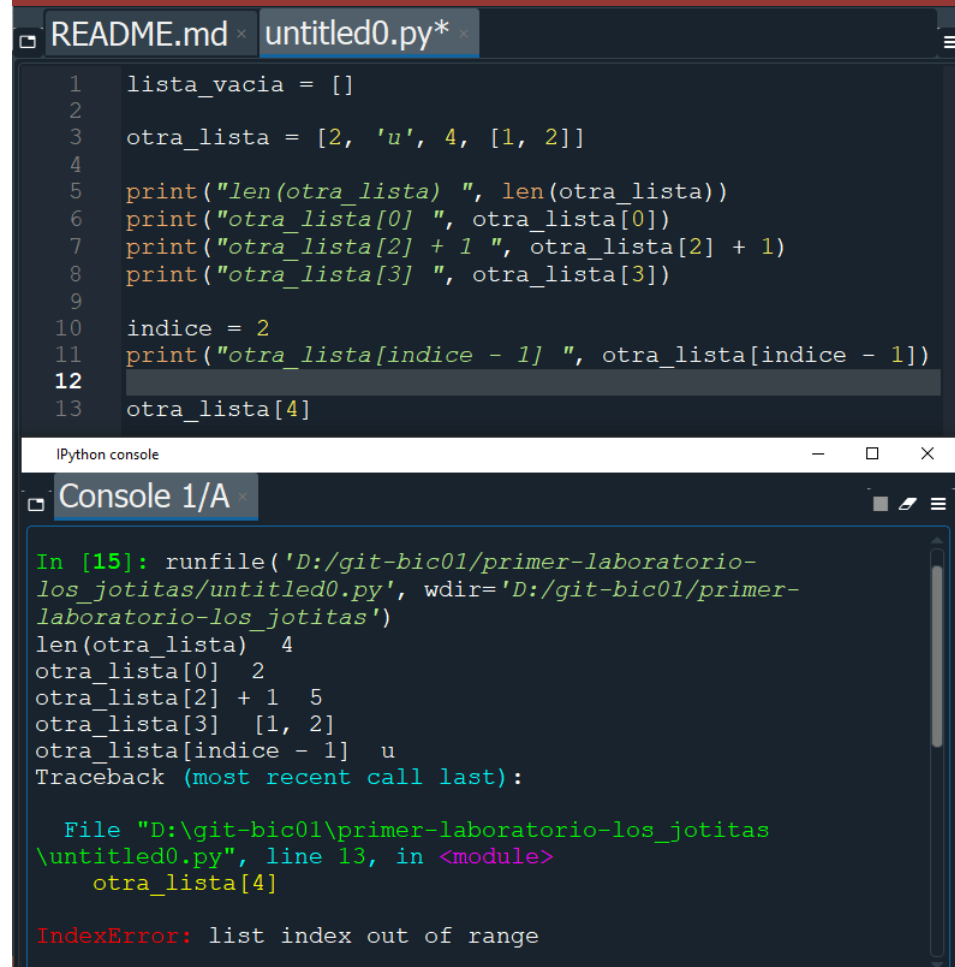


```
tuplas2.py x
1  def procesar_notas(notas_por_alumno):
2      notas = ()
3      alumnos = ()
4      for nota_y_alumno in notas_por_alumno:
5          nota = nota_y_alumno[0]
6          alumno = nota_y_alumno[1]
7          # only add words haven't added before
8          if alumno not in alumnos:
9              notas = notas + (nota,)
10             alumnos = alumnos + (alumno,)
11     minima_nota = min(notas)
12     maxima_nota = max(notas)
13     numero_alumnos = len(alumnos)
14     return (minima_nota, maxima_nota, numero_alumnos)
15
16 notas_parcial = ((6, "Juan"), (20, "Nestor"), (20, "Luis"),
17                  (7, "Jose"), (8, "tomas"), (20, "Jaime"))
18 print(procesar_notas(notas_parcial))

Run:  tuplas2 x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scr
(6, 20, 6)
```

Listas

- Una secuencia ordenada de elementos, accesible mediante índices.
- **Por lo general** contienen elementos del mismo tipo. Esto no es obligatorio.
- Son **mutables**. Los elementos de una lista pueden modificarse.
- Se representan mediante corchetes [].



The screenshot shows a Python IDE with two tabs: 'README.md' and 'untitled0.py*'. The 'untitled0.py' tab is active, displaying the following code:

```
1 lista_vacia = []
2
3 otra_lista = [2, 'u', 4, [1, 2]]
4
5 print("len(otra_lista) ", len(otra_lista))
6 print("otra_lista[0] ", otra_lista[0])
7 print("otra_lista[2] + 1 ", otra_lista[2] + 1)
8 print("otra_lista[3] ", otra_lista[3])
9
10 indice = 2
11 print("otra_lista[indice - 1] ", otra_lista[indice - 1])
12
13 otra_lista[4]
```

Below the code editor is an 'IPython console' window with the title 'Console 1/A'. It shows the output of running the script:

```
In [15]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/primer-
laboratorio-los_jotitas')
len(otra_lista) 4
otra_lista[0] 2
otra_lista[2] + 1 5
otra_lista[3] [1, 2]
otra_lista[indice - 1] u
Traceback (most recent call last):

  File "D:\git-bic01\primer-laboratorio-los_jotitas
\untitled0.py", line 13, in <module>
    otra_lista[4]

IndexError: list index out of range
```


Modificando Elementos

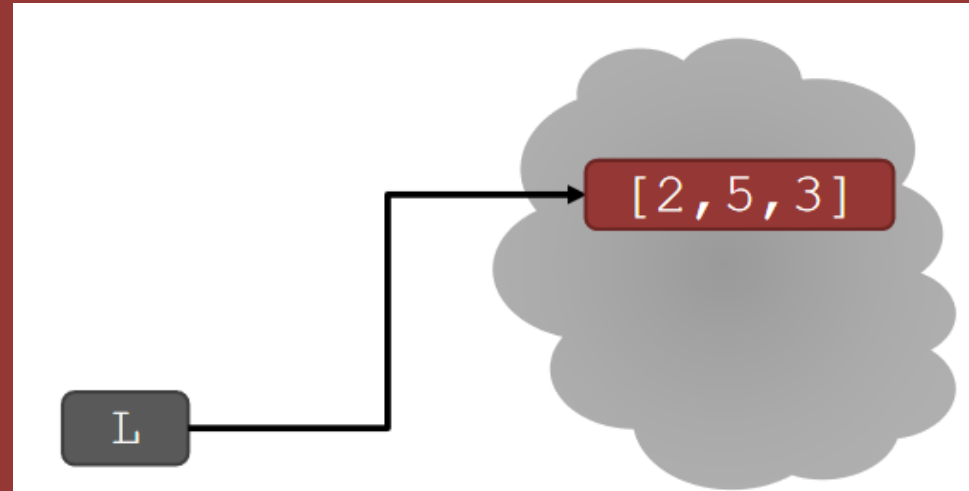
```
1 L = [2, 1, 3]
2 print("L original: ", L)
3
4 L[1] = 5
5 print("L modificada: ", L)
6
```

IPython console

Console 1/A

```
In [18]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
L original: [2, 1, 3]
L modificada: [2, 5, 3]
```

```
In [19]:
```



Iterando sobre Listas

- Las dos técnicas producen el mismo resultado.
- `range(n)` produce valores de 0 a $n - 1$.
- En una lista `L` los índices van desde 0 a $\text{len}(L) - 1$.

```
1 impuestos_por_mes = [100.0, 200.0, 100.0]
2
3 total_impuestos = 0.0
4 for indice in range(len(impuestos_por_mes)):
5     total_impuestos += impuestos_por_mes[indice]
6
7 print("total_impuestos: ", total_impuestos)
8
9 total_impuestos = 0.0
10 for impuesto_mensual in impuestos_por_mes:
11     total_impuestos += impuesto_mensual
12
13 print("total_impuestos: ", total_impuestos)
14
15
```

IPython console

Console 1/A

```
In [21]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/primer-
laboratorio-los_jotitas')
total_impuestos: 400.0
total_impuestos: 400.0
```

Agregar Elementos

- Para agregar el elemento `e` a la lista `l`, usar `l.append(e)`. Esta operación **muta** la lista.
- Nota al pie:
 - ◆ En Python, todos son objetos. Incluso las listas.
 - ◆ Los objetos tienen asociados datos y funciones (llamados métodos).
 - ◆ Para acceder a ellos, usamos `objeto.haz_algo()`

```
1 impuestos_por_mes = [100.0, 200.0, 100.0]
2 print("impuestos_por_mes ", impuestos_por_mes)
3
4 impuestos_por_mes.append(150.0)
5 print("impuestos_por_mes ", impuestos_por_mes)
6
7
```

IPython console

Console 1/A

```
In [24]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
impuestos_por_mes  [100.0, 200.0, 100.0]
impuestos_por_mes  [100.0, 200.0, 100.0, 150.0]
```

```
In [25]: |
```

Concatenar Listas

- El operador + se puede usar para concatenar , o combinar, listas. Este genera una nueva lista.
- Para mutar una lista L, podemos usar `L.extend(otra_lista)`.

```
1 una_lista = [2, 1, 3]
2 otra_lista = [4, 5, 6]
3
4 lista_concatenada = una_lista + otra_lista
5 print("lista_concatenada:", lista_concatenada)
6
7 print("Antes de extend:", una_lista)
8 una_lista.extend([0, 6])
9 print("Despues de extend:", una_lista)
```

IPython console

Console 1/A

```
In [30]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
lista_concatenada: [2, 1, 3, 4, 5, 6]
Antes de extend: [2, 1, 3]
Despues de extend: [2, 1, 3, 0, 6]
```

Eliminar Elementos

- Para eliminar un elemento en el índice i de la lista L , usar `del(L[i])`.
- Para **obtener y eliminar** el último elemento de la lista L , usar `L.pop()`.
- Para eliminar un elemento cuyo valor es v de la lista L , usar `L.remove(v)`.
 - ◆ Si hay varios elementos, elimina sólo el primero.
 - ◆ Si no hay elementos, se produce un error.

```
1 una_lista = [2, 1, 3, 6, 3, 7, 0]
2
3 una_lista.remove(2)
4 print("una_lista ", una_lista)
5
6 una_lista.remove(3)
7 print("una_lista ", una_lista)
8
9 del(un_a_lista[1])
10 print("una_lista ", una_lista)
11
12 ultimo_elemento = una_lista.pop()
13 print("ultimo_elemento ", ultimo_elemento)
14 print("una_lista ", una_lista)
15
```

IPython console

Console 1/A

```
In [35]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
una_lista [1, 3, 6, 3, 7, 0]
una_lista [1, 6, 3, 7, 0]
una_lista [1, 3, 7, 0]
ultimo_elemento 0
una_lista [1, 3, 7]
```

Listas y Cadenas

- Dada una cadena `s`, `list(s)` produce una lista con todos los caracteres en `s`.
- Dada una cadena `s` y un carácter `c`, `s.split(c)` divide a `s` usando `c` como divisor. Si se omite `c`, se divide en base a espacios en blanco.
- Dado un carácter `c` y una lista de caracteres `L`, `c.join(L)` produce una cadena usando los elementos de `L` con `c` entre cada par de caracteres.

```
1 una_cadena = "Yo <3 FIIS"
2
3 print(list(una_cadena))
4 print(una_cadena.split("<"))
5
6 una_lista = ["U", "N", "I"]
7
8 print("".join(una_lista))
9 print("_".join(una_lista))
```

IPython console

Console 1/A

```
In [38]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
['Y', 'o', ' ', '<', '3', ' ', 'F', 'I', 'I', 'S']
['Yo ', '3 FIIS']
UNI
U_N_I
```

Otras Operaciones en Listas

- `sorted(L)` produce una nueva versión ordenada de `L`, sin mutar `L`.
- `L.sort()` muta `L`, ordenando sus elementos.
- `L.reverse()` muta `L`, ordenando sus elementos en reversa.

```
1  una_lista = [9, 6, 0, 3]
2  lista_ordenada = sorted(un_a_lista)
3
4  print("una_lista ", una_lista)
5  print("lista_ordenada ", lista_ordenada)
6
7  una_lista.sort()
8  print("una_lista ", una_lista)
9
10 una_lista.reverse()
11 print("una_lista ", una_lista)
12
```

IPython console

Console 1/A

```
In [41]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/
primer-laboratorio-los_jotitas')
una_lista  [9, 6, 0, 3]
lista_ordenada  [0, 3, 6, 9]
una_lista  [0, 3, 6, 9]
una_lista  [9, 6, 3, 0]
```



- Las listas son objetos mutables almacenados en memoria.
- Tenemos acceso a estos objetos mediante una o más variables.
- Al modificar un objeto mediante una variable, estos cambios afectarán a todas las variables que accedan a este objeto.
- Cuando trabajamos con objetos mutables, debemos tener cuidado con potenciales efectos secundarios.



←

→

↺

🏠

python tutor.com/visualize.html#mode=display

[Get live help](#) for free in the [Python tutoring Discord](#) chat room

Python 3.6
([known limitations](#))

```
1 universidades = ["UNI", "PUCP", "UNMSM"]
2 casas_de_estudio = universidades
3
4 casas_de_estudio.append("UNALM")
5
6 print("universidades: ", universidades)
→ 7 print("casas_de_estudio: ", casas_de_estudio)
```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First

< Prev

Next >

Last >>

Done running (5 steps)

[Customize visualization](#) (NEW!)

Print output (drag lower right corner to resize)

universidades: ['UNI', 'PUCP', 'UNMSM', 'UNALM']
casas_de_estudio: ['UNI', 'PUCP', 'UNMSM', 'UNALM']

Frames

Objects

Global frame

universidades

casas_de_estudio

list

0	1	2	3
"UNI"	"PUCP"	"UNMSM"	"UNALM"

Clonando una Lista



← → ↺ 🏠 python tutor.com/visualize.html#mode=display

[Get live help](#) for free in the [Python tutoring Discord](#) chat room

Python 3.6
([known limitations](#))

```
1 universidades_publicas = ["UNI", "UNMSM", "UNALM"]
2
3 universidades = universidades_publicas[:]
4 universidades.append("PUCP")
5
6 print("universidades_publicas:", universidades_publicas)
→ 7 print("universidades:", universidades)
```

[Edit this code](#)

→ line that just executed
→ next line to execute

Done running (5 steps)

[Customize visualization](#) (NEW!)

Print output (drag lower right corner to resize)

```
universidades_publicas: ['UNI', 'UNMSM', 'UNALM']
universidades: ['UNI', 'UNMSM', 'UNALM', 'PUCP']
```

Frames Objects

Global frame

universidades_publicas → list

0	1	2
"UNI"	"UNMSM"	"UNALM"

universidades → list

0	1	2	3
"UNI"	"UNMSM"	"UNALM"	"PUCP"

Modificaciones durante iteración

- No lo hagan.
- En la línea 2, Python utiliza un contador interno para recorrer los elementos de la lista.
- Al modificar la lista en la línea 4, el contador no es actualizado.
- Eso ocasiona que se ignore el valor 2.

```
1 def remover_duplicados(una_lista, otra_lista):
2     for un_elemento in una_lista:
3         if un_elemento in otra_lista:
4             una_lista.remove(un_elemento)
5
6 def remover_duplicados_v2(una_lista, otra_lista):
7     copia_una_lista = una_lista[:]
8     for un_elemento in copia_una_lista:
9         if un_elemento in otra_lista:
10             una_lista.remove(un_elemento)
11
12 lista_1 = [1, 2, 3, 4]
13 lista_2 = [1, 2, 5, 6]
14 remover_duplicados(lista_1, lista_2)
15 print(lista_1)
16
17 lista_1 = [1, 2, 3, 4]
18 lista_2 = [1, 2, 5, 6]
19 remover_duplicados_v2(lista_1, lista_2)
20 print(lista_1)
```

Python console

Console 1/A

```
In [46]: runfile('D:/git-bic01/primer-laboratorio-
los_jotitas/untitled0.py', wdir='D:/git-bic01/primer-
laboratorio-los_jotitas')
[2, 3, 4]
[3, 4]
```

AGENDA

1. INTRODUCCIÓN
2. ESTRUCTURAS DE CONTROL
3. CADENAS
4. FUNCIONES
5. TUPLAS, LISTAS, ALIASING, MUTABILIDAD Y CLONES
6. **RECURSIVIDAD Y DICCIONARIOS**
7. EXCEPCIONES



AGENDA

1. INTRODUCCIÓN
2. ESTRUCTURAS DE CONTROL
3. CADENAS
4. FUNCIONES
5. TUPLAS, LISTAS, ALIASING, MUTABILIDAD Y CLONES
6. **RECURSIVIDAD Y DICCIONARIOS**
 - **Recursividad**
 - Dicionarios
7. EXCEPCIONES





→ Algorítmicamente:

- ◆ Un paradigma de diseño “divide y vencerás”: Dividimos un problema en versiones más pequeñas del mismo problema.

→ Semánticamente:

- ◆ Una técnica de programación donde una función se invoca a sí misma.
- ◆ La recursividad no debe ser infinita. Para esto:
 - Debemos contar con uno más casos base, fáciles de resolver.
 - Debemos resolver el mismo problema múltiples veces. Cada vez, el problema se hace más sencillo.

Solución Iterativa

- El uso de bucles como `for` y `while` producen soluciones iterativas.
- Requieren el uso de **variables de estado**, que se actualizan en cada iteración del bucle.
 - ◆ `contador`, desde `otro_factor` hasta 0.
 - ◆ `resultado`, la solución al problema.

```
1 def multiplicar_iterativo(factor, otro_factor):
2     resultado = 0
3
4     contador = otro_factor
5     while contador > 0:
6         resultado += factor
7         contador -= 1
8
9     return resultado
10
11 print(multiplicar_iterativo(8, 7))
```

IPython console

Console 1/A ×

```
In [8]: runfile('C:/Users/cgavi/.spyder-py3/temp.py',
56      Users/cgavi/.spyder-py3')
```

```
In [9]: |
```



$$a * b = \underbrace{a + a + a + a + \dots + a}_{b \text{ times}}$$

b times

$$= a + \underbrace{a + a + a + \dots + a}_{b-1 \text{ times}}$$

b-1 times

$$= a + \boxed{a * (b-1)}$$

*recursive
reduction*

Solución Recursiva

→ Llamada recurrente:

- ◆ Reduce el problema original a una versión **más sencilla/pequeña** de sí mismo.

→ Caso base:

- ◆ Un caso sencillo que puede ser **resuelto directamente**. El problema original se reducirá hasta este valor.

```
MultiplicacionRecursiva.py x
def multiplicacion(a, b):
    # caso base
    if b == 1:
        return a
    # llamada recurrente
    else:
        return a + multiplicacion(a, b - 1)

print(multiplicacion(8, 7))

run: MultiplicacionRecursiva x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\
56
Process finished with exit code 0
```

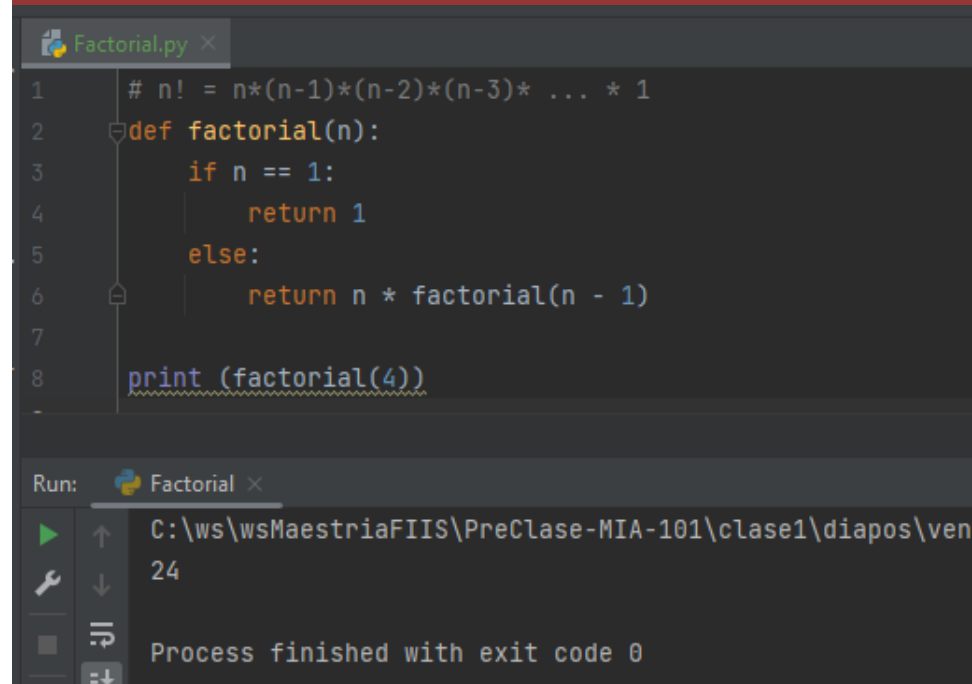
Factoriales

→ Caso base:

- ◆ Por definición, el factorial de 1 es 1.

→ Llamada recurrente:

- ◆ Reducimos el problema hasta llegar al caso base.

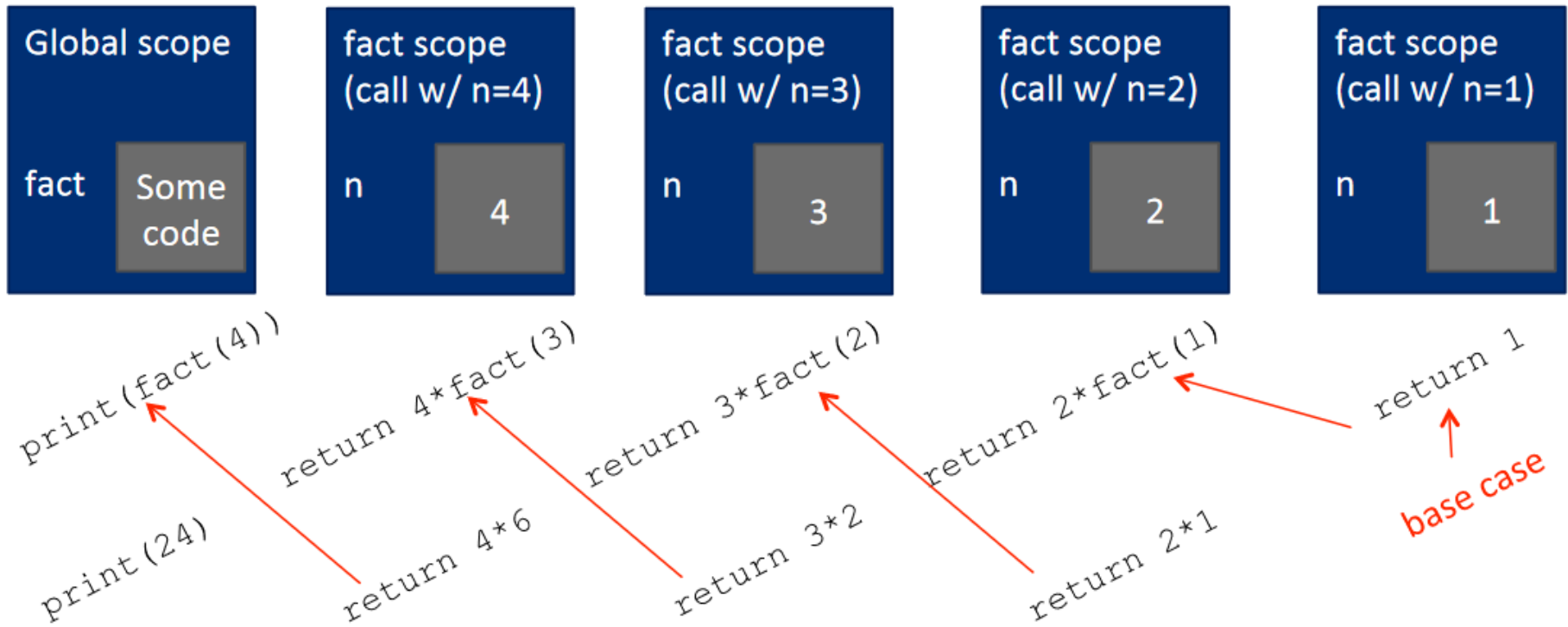


The screenshot shows a Python IDE with a file named 'Factorial.py'. The code defines a recursive function 'factorial(n)' with a base case for n == 1 and a recursive case for n > 1. It also includes a call to 'print(factorial(4))'. Below the code editor, the 'Run' output shows the execution path and the final result, 24, followed by the message 'Process finished with exit code 0'.

```
Factorial.py ×
1 # n! = n*(n-1)*(n-2)*(n-3)* ... * 1
2 def factorial(n):
3     if n == 1:
4         return 1
5     else:
6         return n * factorial(n - 1)
7
8 print(factorial(4))
-

Run: Factorial ×
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\ven
24
Process finished with exit code 0
```

Ámbito en Funciones Recursivas

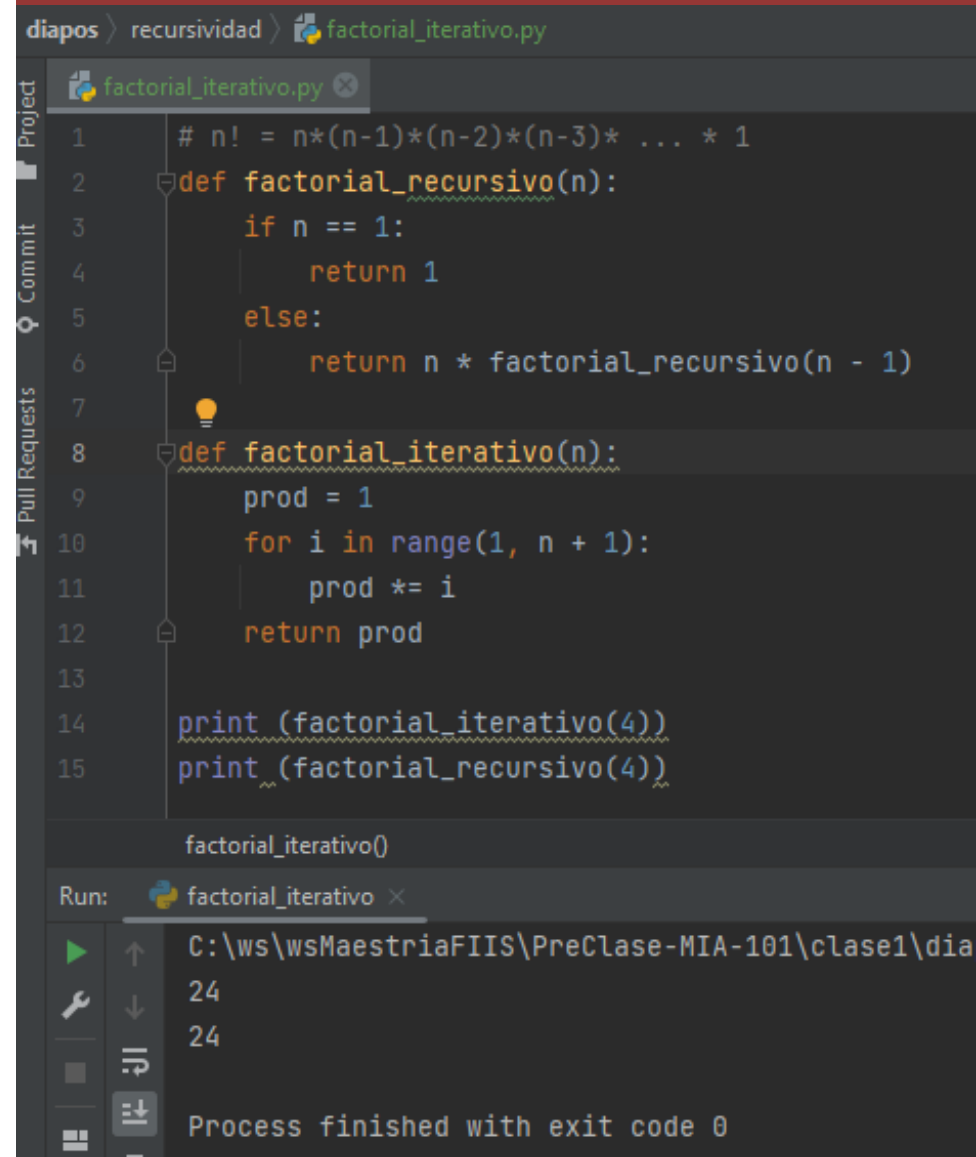




- Cada llamada recurrente genera un nuevo ámbito.
- En cada ámbito, la llamada recurrente no cambia los valores de las variables.
- Una vez que la llamada recurrente termina produciendo un valor, el control regresa al ámbito original.

Iteración y Recursividad

- Las funciones recursivas pueden ser más **sencillas e intuitivas**.
- Para un programador, implementar una función recursiva puede ser **más eficiente**.
- Para una computadora, ejecutar una función recursiva puede ser **ineficiente**.



The screenshot shows a code editor with a file named `factorial_iterativo.py`. The code defines two functions: `factorial_recursivo` and `factorial_iterativo`. The `factorial_recursivo` function uses a recursive approach, while `factorial_iterativo` uses an iterative loop. At the bottom, the code calls both functions with the argument 4. The output window shows the results of these calls, both returning 24, and indicates that the process finished successfully with exit code 0.

```
# n! = n*(n-1)*(n-2)*(n-3)* ... * 1
def factorial_recursivo(n):
    if n == 1:
        return 1
    else:
        return n * factorial_recursivo(n - 1)

def factorial_iterativo(n):
    prod = 1
    for i in range(1, n + 1):
        prod *= i
    return prod

print(factorial_iterativo(4))
print(factorial_recursivo(4))
```

factorial_iterativo()

Run: factorial_iterativo

C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\dia
24
24

Process finished with exit code 0



- Para probar que todos los valores enteros* de n tienen una propiedad:
 - ◆ Primero, probar que valores pequeños de n tienen esta propiedad.
 - $n = 0, n = 1$
 - ◆ Luego probar que, asumiendo que un valor arbitrario n tiene esta propiedad, se concluye que $n + 1$ también tiene esta propiedad.



→ Propiedad: $P(n) = 1 + 2 + \dots + n = n * (n + 1) / 2$

→ Prueba:

◆ $P(1) = 1$. De acuerdo a $P(1) = 1 * (1 + 1) / 2$.

- $P(1)$ es verdadera.

◆ Es $P(k + 1) = (k + 1) * (k + 2) / 2$ verdadera?

- $P(k + 1) = (1 + \dots + k) + (k + 1)$

- Asumiendo $P(k)$ es verdadera:

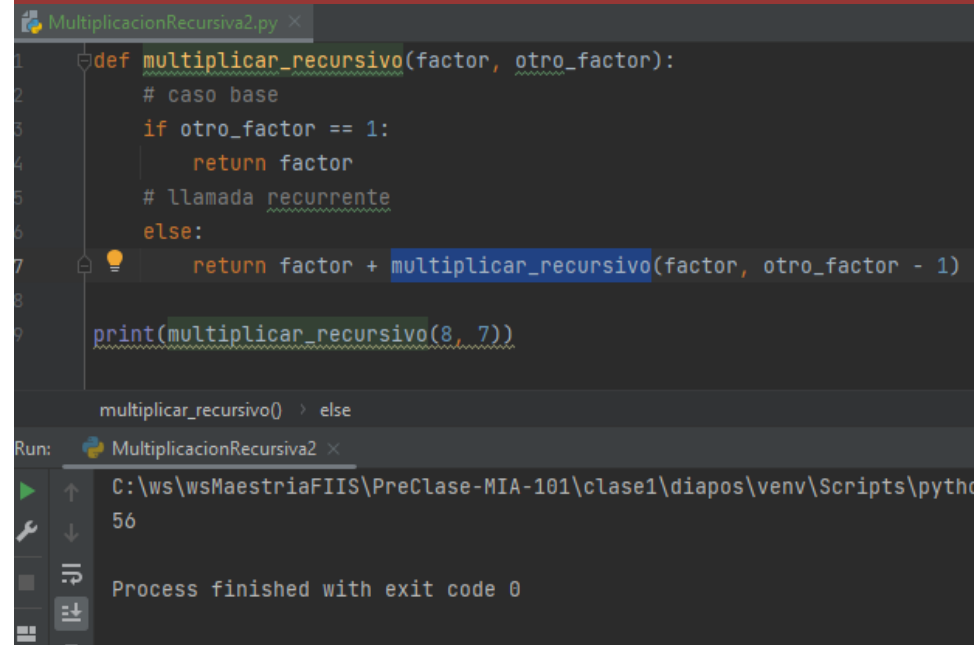
- $P(k + 1) = k * (k + 1) / 2 + k + 1$

- $P(k + 1) = (k + 1) * (k + 2) / 2$

◆ $P(n)$ es verdadera para todos los valores de $n \geq 0$.

En Código

- En el caso base ,
multiplicar_recursivo
devuelve el valor correcto.
- En la llamada recurrente, si
asumimos que
multiplicar_recursivo
funciona para valores menores
que otro_factor, entonces
funciona también para
otro_factor.
- Entonces, por inducción
multiplicar_recursivo
funciona correctamente.



```
def multiplicar_recursivo(factor, otro_factor):  
    # caso base  
    if otro_factor == 1:  
        return factor  
    # llamada recurrente  
    else:  
        return factor + multiplicar_recursivo(factor, otro_factor - 1)  
  
print(multiplicar_recursivo(8, 7))
```

Run: MultiplicacionRecursiva2 x

C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe
56
Process finished with exit code 0



- En un templo de la India, existe una gran sala con 3 postes.
- Un poste está rodeado por 64 discos dorados.
- Los sacerdotes mueven los discos de acuerdo a las reglas de Brahma:
 - ◆ Los discos se mueven de uno en uno.
 - ◆ No se puede colocar un disco grande encima de uno más pequeño.
- Cuando muevan todos los discos al segundo poste, el mundo terminará.



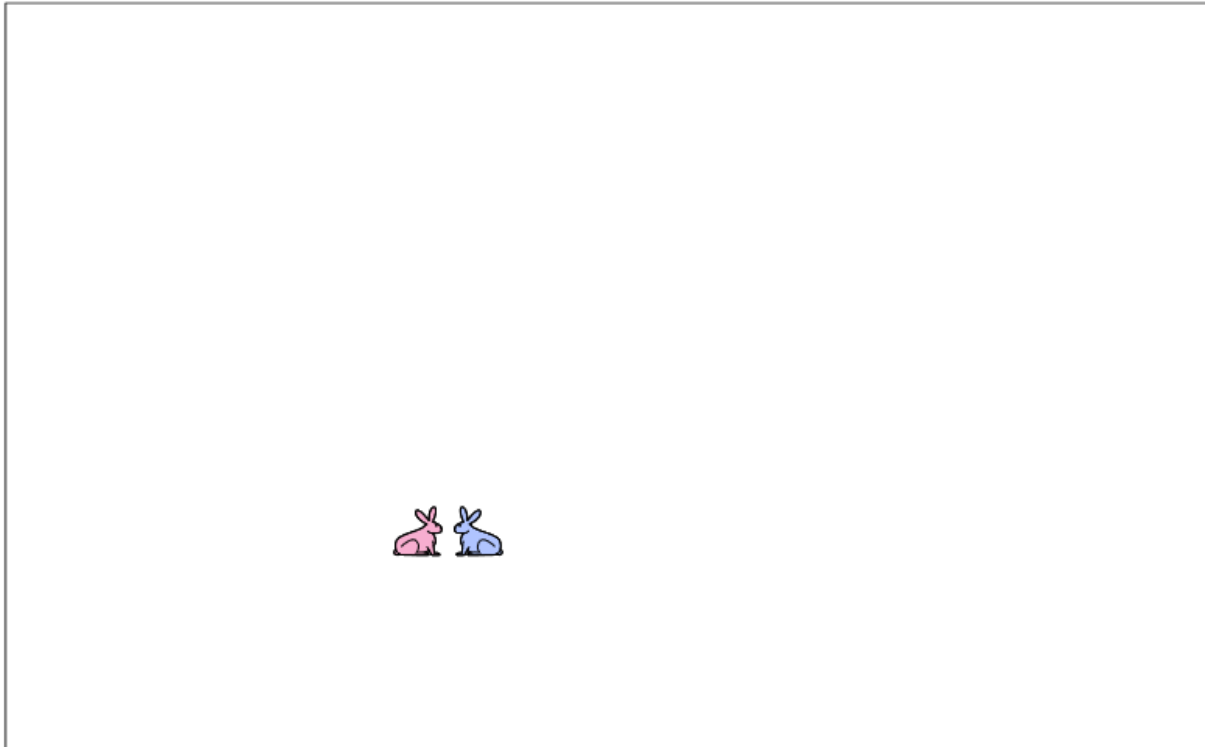
Dada a conocer por Leonardo de Pisa (alias Fibonacci) en 1202:

1. Empezamos con una pareja de conejos en un corral.
2. Al llegar al mes de edad, los conejos se cruzan.
3. El embarazo de los conejos dura un mes.
4. Asumiendo conejos inmortales; y que cada pareja de conejos da a luz a una pareja (macho y hembra) cada mes a partir del segundo mes.

¿Cuántas parejas tenemos al final de un año?

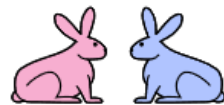


Inicio del Primer Mes = 1



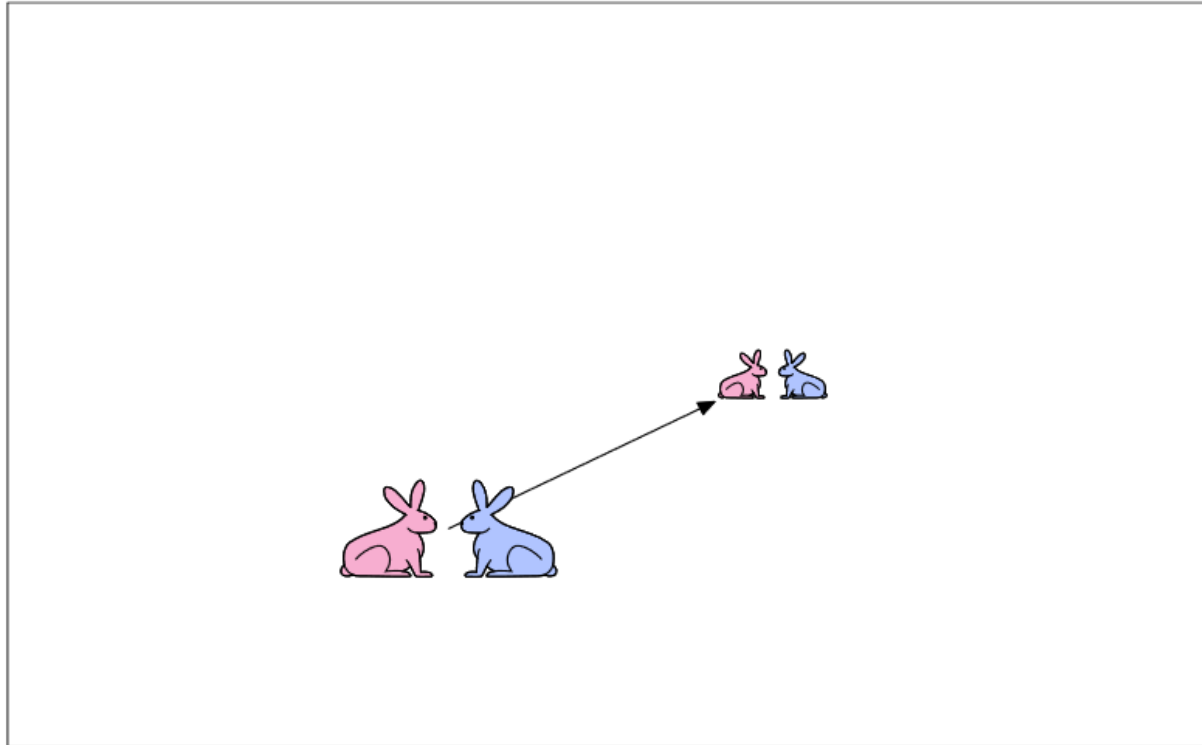
Demo courtesy of Prof. Denny Freeman and Adam Hartz

Fin del Primer Mes = 1



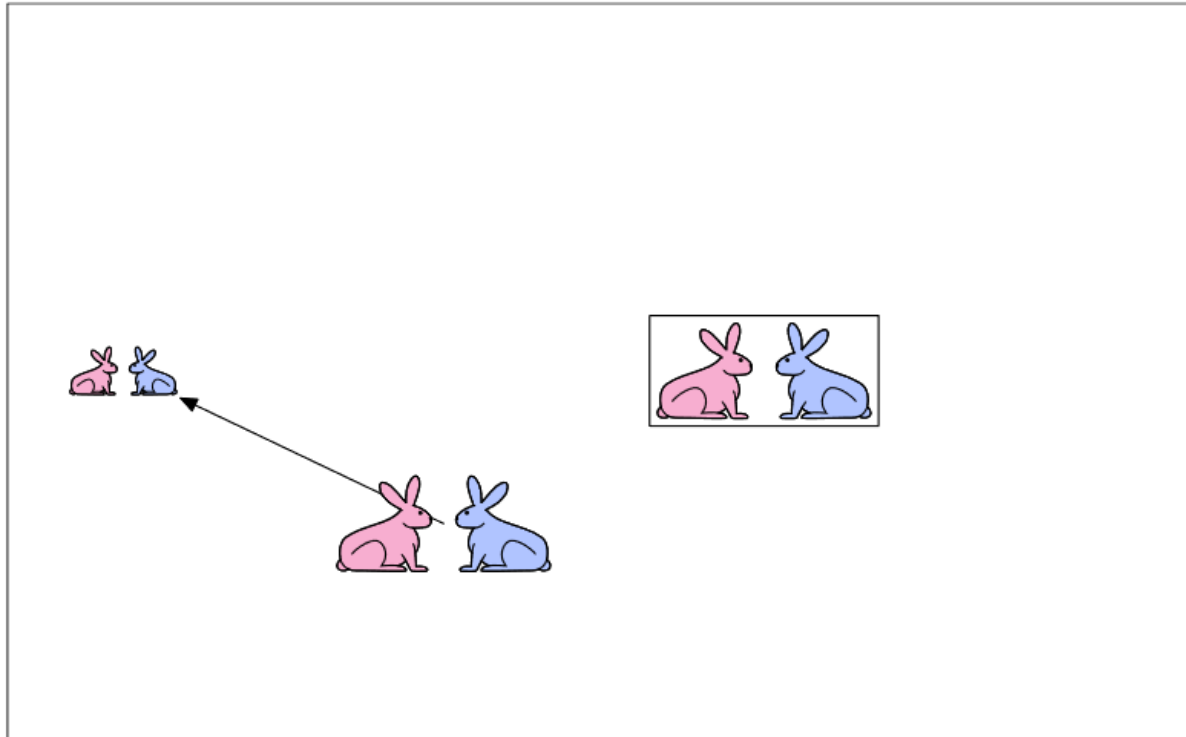
Demo courtesy of Prof. Denny Freeman and Adam Hartz

Fin del Segundo Mes = 2



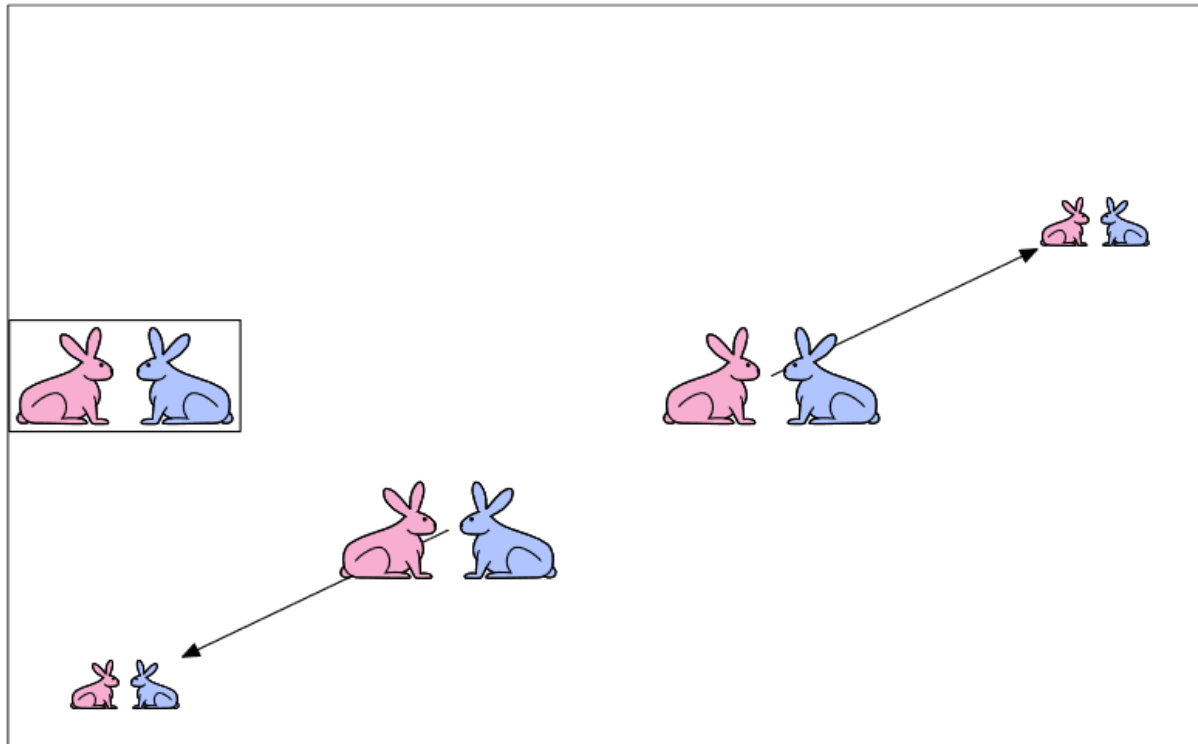
Demo courtesy of Prof. Denny Freeman and Adam Hartz

Fin del Tercer Mes = 3



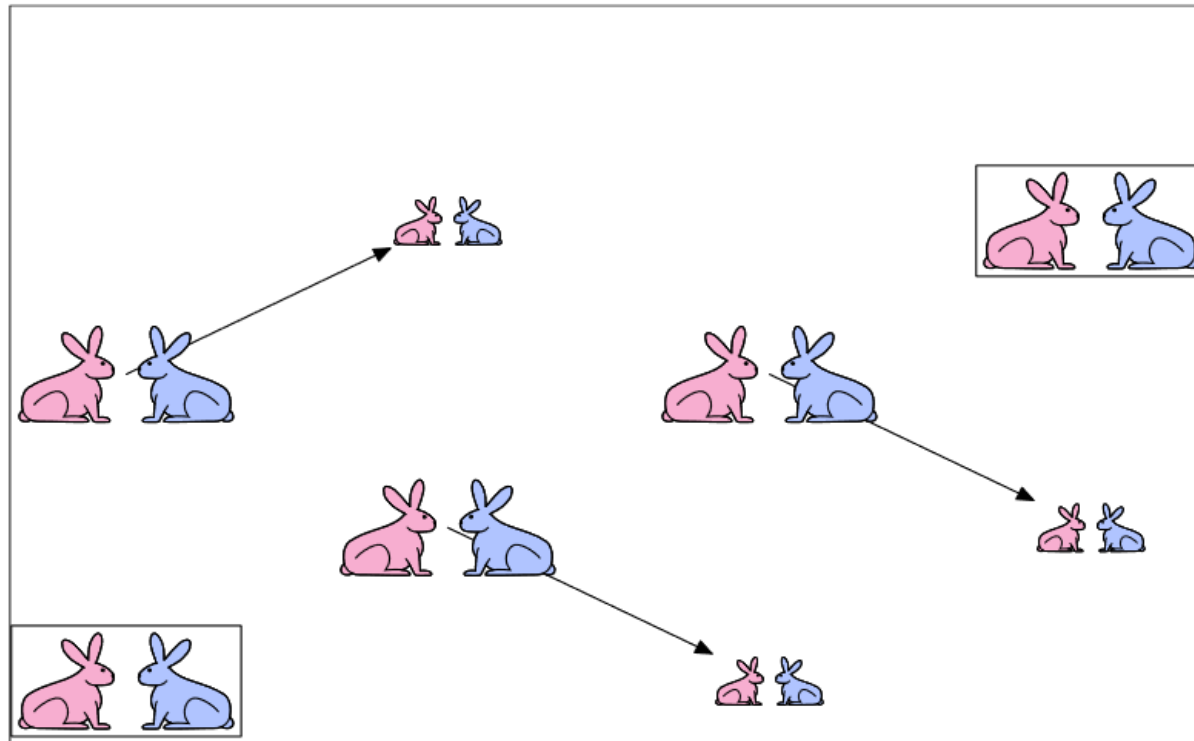
Demo courtesy of Prof. Denny Freeman and Adam Hartz

Fin del Cuarto Mes = 5



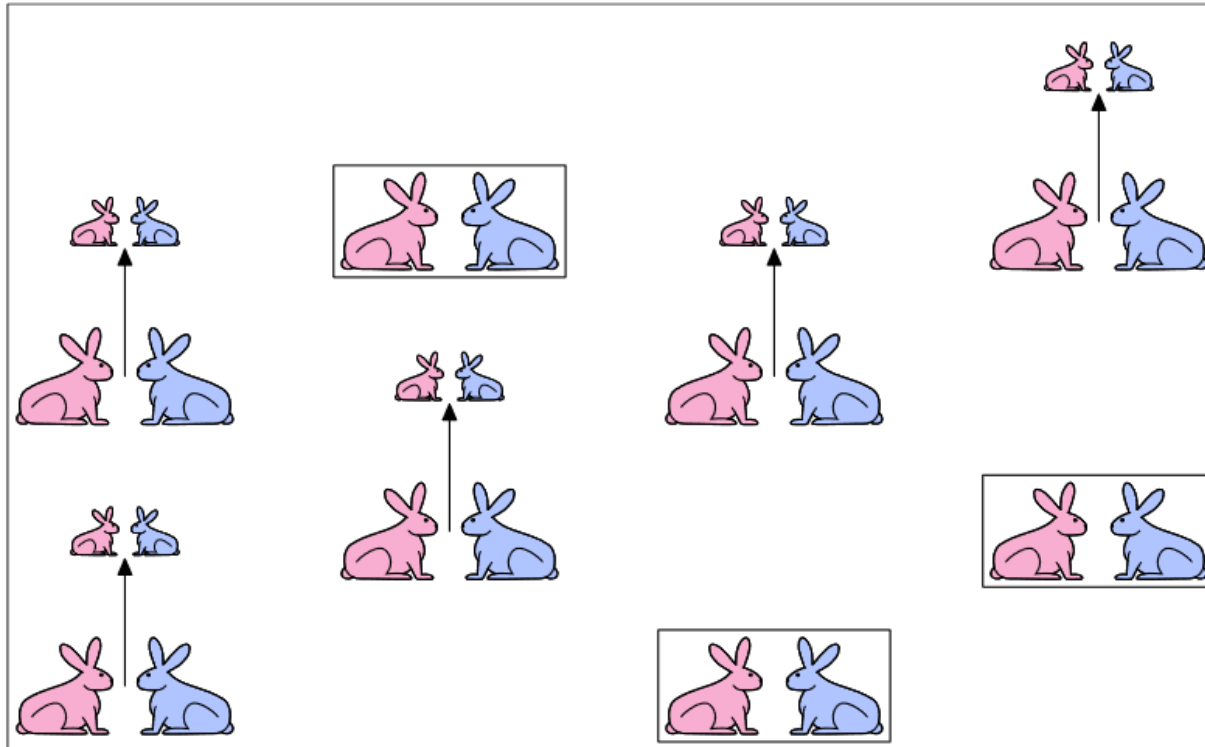
Demo courtesy of Prof. Denny Freeman and Adam Hartz

Fin del Quinto Mes = 8



Demo courtesy of Prof. Denny Freeman and Adam Hartz

Fin del Sexto Mes = 13



Demo courtesy of Prof. Denny Freeman and Adam Hartz

Secuencia de Fibonacci

→ $\text{parejas}(n) = \text{parejas}(n - 1) + \text{parejas}(n - 2)$

- ◆ Cada pareja viva en $n-2$ producirá una nueva pareja en n
- ◆ Adicionalmente a las parejas vivas en $n-1$

Serie de Fibonacci	1	1	2	3	5	8	13	21	...
	0	1	2	3	4	5	6	6	

```
fibonacci.py x
1 def fibonacci(x):
2     """
3     asume que x es un int >= 0
4     returns Fibonacci de x
5     """
6     if x == 0 or x == 1:
7         return 1
8     else:
9         return fibonacci(x - 1) + fibonacci(x - 2)
10
11 for numero in range(7):
12     print("Fibonacci de " + str(numero) + ": " + str(fibonacci(numero)))

for numero in range(7)

Run: fibonacci x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.
Fibonacci de 0: 1
Fibonacci de 1: 1
Fibonacci de 2: 2
Fibonacci de 3: 3
Fibonacci de 4: 5
Fibonacci de 5: 8
Fibonacci de 6: 13
```

Amor Azul

Ramera, de todo te di,

Mariposa colosal, sí,

yo de todo te di.

Poda la rosa, Venus.

El átomo como tal

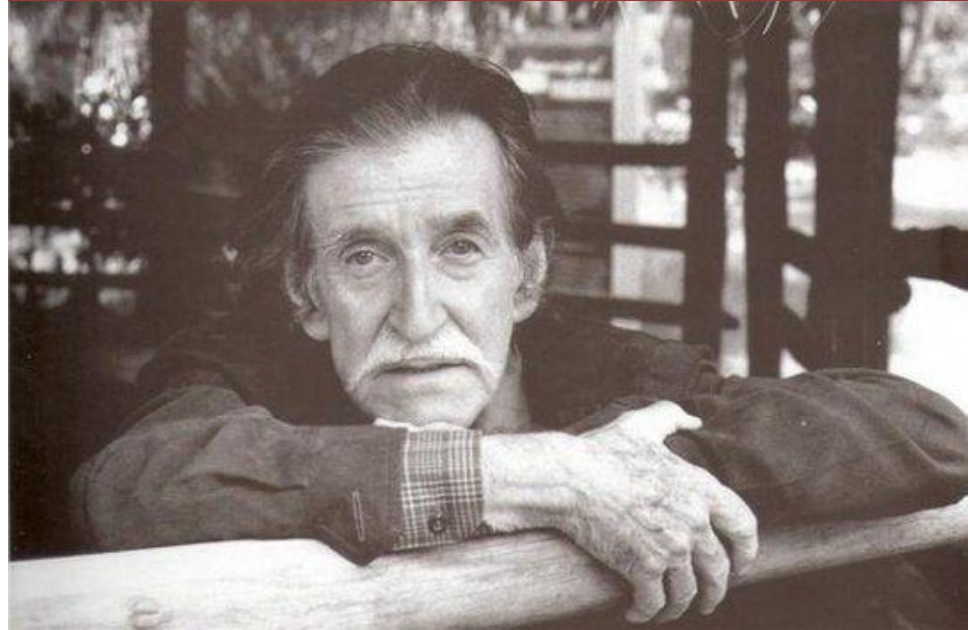
es un evasor alado.

Pide, todo te doy: isla,

sol, ocaso, pirámide.

Todo te daré: mar, luz, aroma.

Dario Lancini (1932 - 2010)





Un palíndromo es una cadena de caracteres que se lee igual en un sentido o en otro.

1. Pre-procesar: Eliminar signos de puntuación, espacios en blanco, y convertir a minúsculas.
2. Caso base: Una cadena con 0 o 1 caracter es un palíndromo.
3. Llamada recurrente: Si el primer y el último carácter son iguales, es un palíndromo si los caracteres del medio son también un palíndromo.

Solución Recursiva

→ `pre_procesar("Yo hago yoga hoy.")`

◆ Resultado:

- `"yohagoyogahoy"`

→ `evaluar("yohagoyogahoy")`

◆ Dos condiciones:

- `"y" == "y"`
- `evaluar("ohagoyogaho")`

```
palindromo.py x
1  def es_palindromo(frase):
2
3      def pre_procesar(frase):
4          frase = frase.lower()
5          resultado = ''
6          for letra in frase:
7              if letra in 'abcdefghijklmnopqrstuvwxyz':
8                  resultado = resultado + letra
9          return resultado
10
11     def evaluar(frase):
12         if len(frase) <= 1:
13             return True
14         else:
15             return frase[0] == frase[-1] and evaluar(frase[1:-1])
16
17     return evaluar(pre_procesar(frase))
18
19 print(es_palindromo("Yo hago yoga hoy."))

es_palindromo()
Run: palindromo x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\pyt
True
```

AGENDA

1. INTRODUCCIÓN
2. ESTRUCTURAS DE CONTROL
3. CADENAS
4. FUNCIONES
5. TUPLAS, LISTAS, ALIASING, MUTABILIDAD Y CLONES
6. **RECURSIVIDAD Y DICCIONARIOS**
 - Recursividad
 - **Diccionarios**
7. EXCEPCIONES

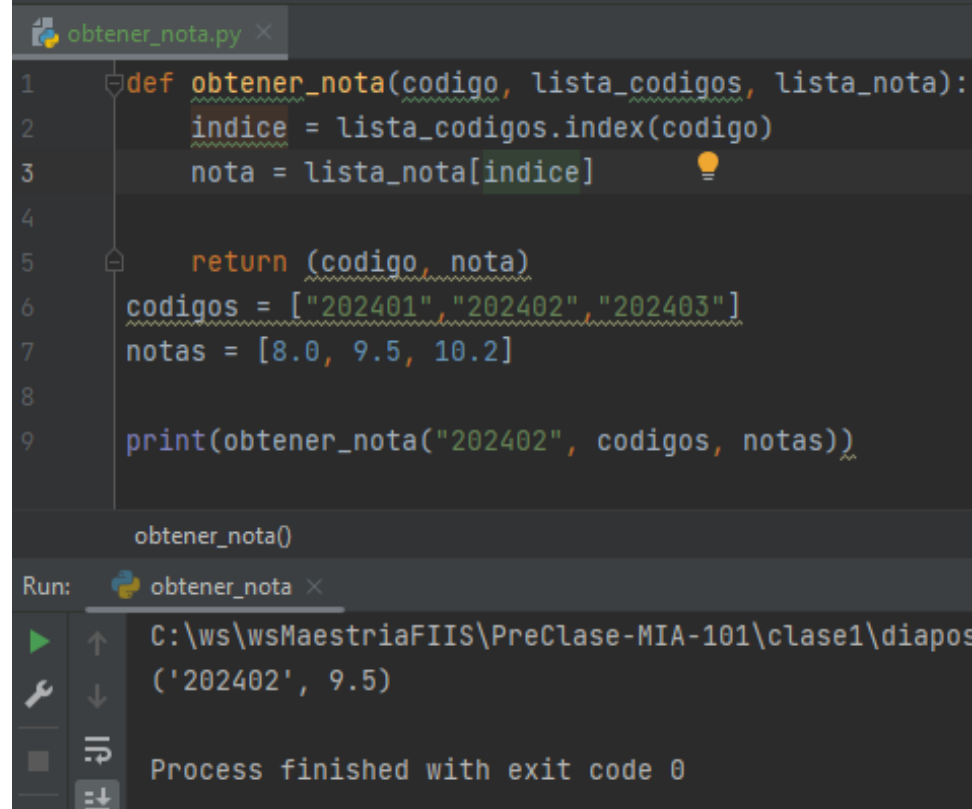


Notas por Alumno

→ Usando listas:

- ◆ Una lista por dato.
- ◆ Todas las listas deben tener el mismo orden y tamaño.
- ◆ Ubicamos información mediante el índice.

→ Requiere mantener múltiples listas.



```
obtener_nota.py x
1 def obtener_nota(codigo, lista_codigos, lista_nota):
2     indice = lista_codigos.index(codigo)
3     nota = lista_nota[indice]
4
5     return (codigo, nota)
6     codigos = ["202401", "202402", "202403"]
7     notas = [8.0, 9.5, 10.2]
8
9     print(obtener_nota("202402", codigos, notas))

obtener_nota()

Run: obtener_nota x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos
('202402', 9.5)

Process finished with exit code 0
```

Diccionarios

- En vez de usar múltiples listas, podemos usar **una sola estructura de datos**.
- Podemos ubicar información **directamente en base al dato de interés**, en vez de usar índices enteros.

A list

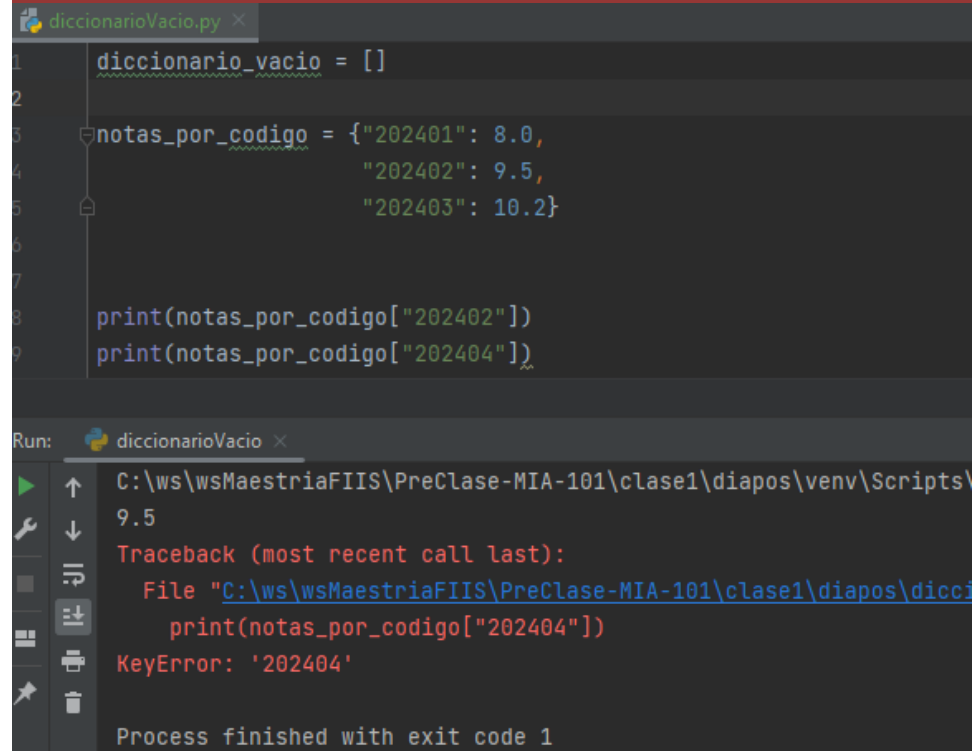
0	Elem 1
1	Elem 2
2	Elem 3
3	Elem 4
...	...

A dictionary

Key 1	Val 1
Key 2	Val 2
Key 3	Val 3
Key 4	Val 4
...	...

Diccionarios en Python

- Almacenan pares conformados por **clave y valor**.
- Para acceder a un valor en base a la clave, usamos corchetes `[]`, como en listas.
- En caso la clave no exista en el diccionario, se produce un error.



The screenshot shows a Python IDE with a file named `diccionarioVacio.py`. The code defines an empty dictionary `diccionario_vacio = {}` and a dictionary `notas_por_codigo` with three entries: `"202401": 8.0`, `"202402": 9.5`, and `"202403": 10.2`. It then prints the value for key `"202402"` (9.5) and attempts to print the value for key `"202404"`. The output shows `9.5` followed by a `KeyError: '202404'` traceback. The process finished with exit code 1.

```
diccionarioVacio.py
1  diccionario_vacio = {}
2
3  notas_por_codigo = {"202401": 8.0,
4                      "202402": 9.5,
5                      "202403": 10.2}
6
7
8  print(notas_por_codigo["202402"])
9  print(notas_por_codigo["202404"])

Run:
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe
9.5
Traceback (most recent call last):
  File "C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\diccionarioVacio.py", line 9, in <module>
    print(notas_por_codigo["202404"])
  KeyError: '202404'

Process finished with exit code 1
```

Operaciones con Diccionarios

- Evaluamos si una clave **existe** en el diccionario mediante `in`.
- **Agregamos** claves y valores mediante corchetes `[]`.
- **Eliminamos** una clave y su respectivo valor con `del ()`.

```
diccionarioVacio2.py x
1 notas_por_codigo = {"202401": 8.0,
2                       "202402": 9.5,
3                       "202403": 10.2}
4
5 print("202404" in notas_por_codigo)
6 notas_por_codigo["202404"] = 18.0
7
8 print("202404" in notas_por_codigo)
9 print(notas_por_codigo["202404"])
10
11 del(notas_por_codigo["202404"])
12 print("202404" in notas_por_codigo)
13
```

Run: diccionarioVacio2 x

↑	C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\
↓	False
↕	True
↕	18.0
↕	False
✖	Process finished with exit code 0

Operaciones con Diccionarios

- `un_diccionario.keys()` produce una secuencias con todas las claves de `un_diccionario`.
- `un_diccionario.values()` produce una secuencias con todos los valores de `un_diccionario`.
- ~~El orden es arbitrario.~~

```
diccionarioVacio3.py x
1 notas_por_codigo = {"202401": 8.0,
2                       "202402": 9.5,
3                       "202403": 10.2}
4
5 print(notas_por_codigo.keys())
6 print(notas_por_codigo.values())

Run: diccionarioVacio3 x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\di
dict_keys(['202401', '202402', '202403'])
dict_values([8.0, 9.5, 10.2])
```

Claves y Valores

Claves

- Deben ser únicas.
- Deben ser de tipos inmutables (enteros, cadenas de caracteres, tuplas, booleanos).
- ~~→ El orden es arbitrario.~~

Valores

- Pueden tener valores duplicados.
- Pueden ser de cualquier tipo, incluyendo listas y diccionarios.
- ~~→ El orden es arbitrario.~~

Listas y Diccionarios

Listas

- Secuencia ordenada de elementos.
- Accedemos a elementos mediante índices.
- Los índices están ordenados.
- Los índices son números enteros

Diccionarios

- Asocia claves a valores.
- Accedemos a valores mediante claves.
- ~~→ No hay garantías respecto al orden.~~
- Las claves pueden ser de cualquier tipo inmutable.

Analizando Letras de Canciones

→ Creamos un diccionario de frecuencias con claves `string` y valores `int`.

```
contarPalabras.py
1 def contar_palabras(lista_palabras):
2     frecuencias = {}
3     for palabra in lista_palabras:
4         if palabra in frecuencias:
5             frecuencias[palabra] += 1
6         else:
7             frecuencias[palabra] = 1
8
9     return frecuencias
10
11 cancion = """
12 si ellos están mintiendo, por favor defiéndete
13 yo sé que no lo harás, pues dicen la verdad
14 es una pena siempre seguirás doliéndome
15 y culpable o no
16 ¿qué le puedo hacer ya?
17 míenteme como siempre
18 por favor míenteme
19 necesito creerte
20 convénceme
21 míenteme con un beso
22 que parezca de amor
23 necesito quererte
24 culpable o no
25 """
26
27 diccionario_frecuencias = contar_palabras(cancion.split())
28 print(diccionario_frecuencias)

Run: contarPalabras (1)
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe C:/ws/wsMaestriaFIIS/Pr
{'si': 1, 'ellos': 1, 'están': 1, 'mintiendo': 1, 'por': 2, 'favor': 2, 'defiéndete': 1, 'yo': 1,
Process finished with exit code 0
```

Palabra más frecuente

- Usamos una lista `palabras_comunes`, en caso se tenga más de una palabra más frecuente.
- El valor de retorno es una tupla, con la lista de palabras `palabras_comunes` y la frecuencia máxima `maxima_frecuencia`

```
todosPalabras.py x
28
29 diccionario_frecuencias = contar_palabras(cancion.split())
30
31 def palabra_mas_comun(frecuencia_por_palabra):
32     frecuencias = frecuencia_por_palabra.values()
33     maxima_frecuencia = max(frecuencias)
34     palabras_comunes = []
35     for palabra in frecuencia_por_palabra:
36         if frecuencia_por_palabra[palabra] == maxima_frecuencia:
37             palabras_comunes.append(palabra)
38
39     return (palabras_comunes, maxima_frecuencia)
40
41 palabras_comunes = palabra_mas_comun(diccionario_frecuencias)
42 print(palabras_comunes)
```

```
Run: todosPalabras x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe
(['no', 'miénteme'], 3)

Process finished with exit code 0
```

Palabra que ocurren al menos n veces

- n es un parámetro.
- El valor de retorno es una lista de tuplas, donde cada tupla está compuesta por un listado de palabras y su frecuencia.
- Podemos utilizar la función `palabra_mas_comun()`.

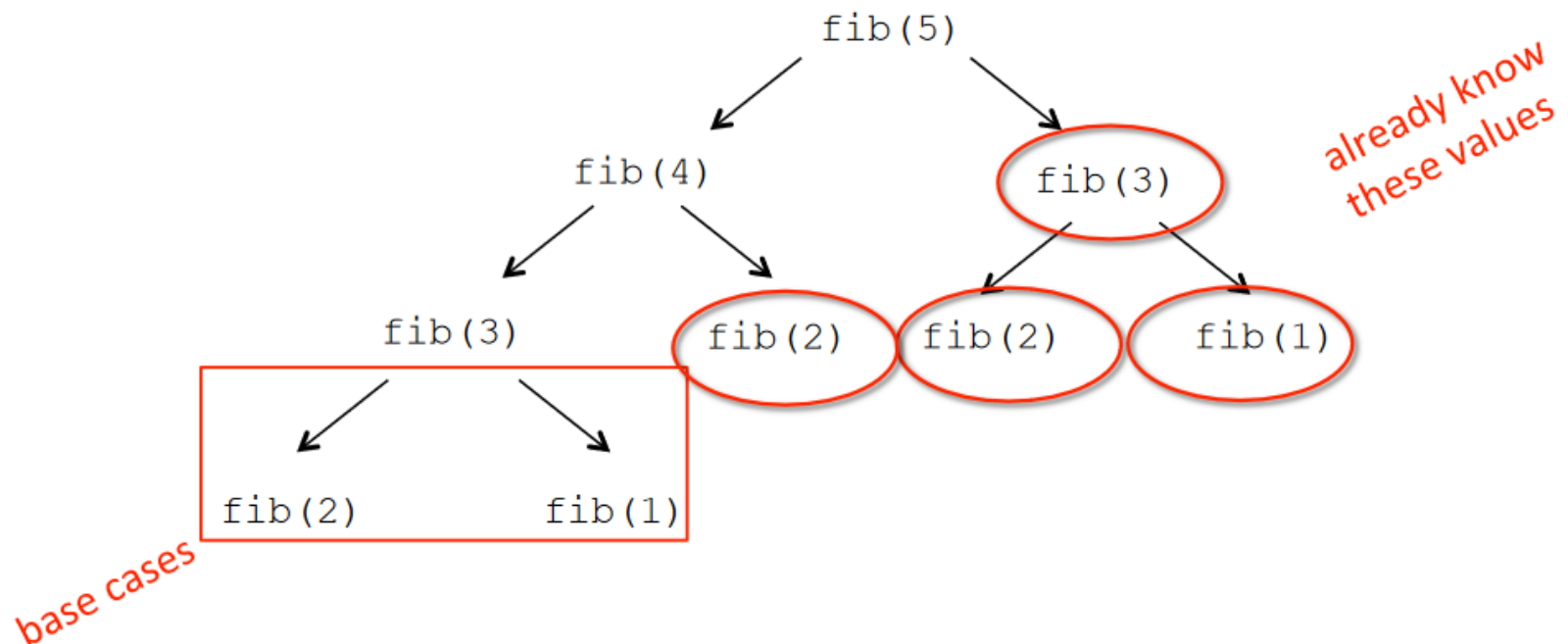
```
todosPalabras.py
44 def palabras_por_frecuencia(frecuencia_por_palabra, frecuencia_minima):
45     resultado = []
46     terminado = False
47     while not terminado:
48         palabra_frecuencia = palabra_mas_comun(frecuencia_por_palabra)
49         if palabra_frecuencia[1] >= frecuencia_minima:
50             resultado.append(palabra_frecuencia)
51             for palabra in palabra_frecuencia[0]:
52                 del(frecuencia_por_palabra[palabra])
53         else:
54             terminado = True
55     return resultado
56
57 al_menos_dos_veces = palabras_por_frecuencia(diccionario_frecuencias, 2)
58 print(al_menos_dos_veces)
59
60 palabras_por_frecuencia()

Run: todosPalabras
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe C:/ws/wsMaestriaFIIS
[[('no', 'miénteme'], 3), ('por', 'favor', 'que', 'siempre', 'culpable', 'o', 'necesito'], 2)]

Process finished with exit code 0
```

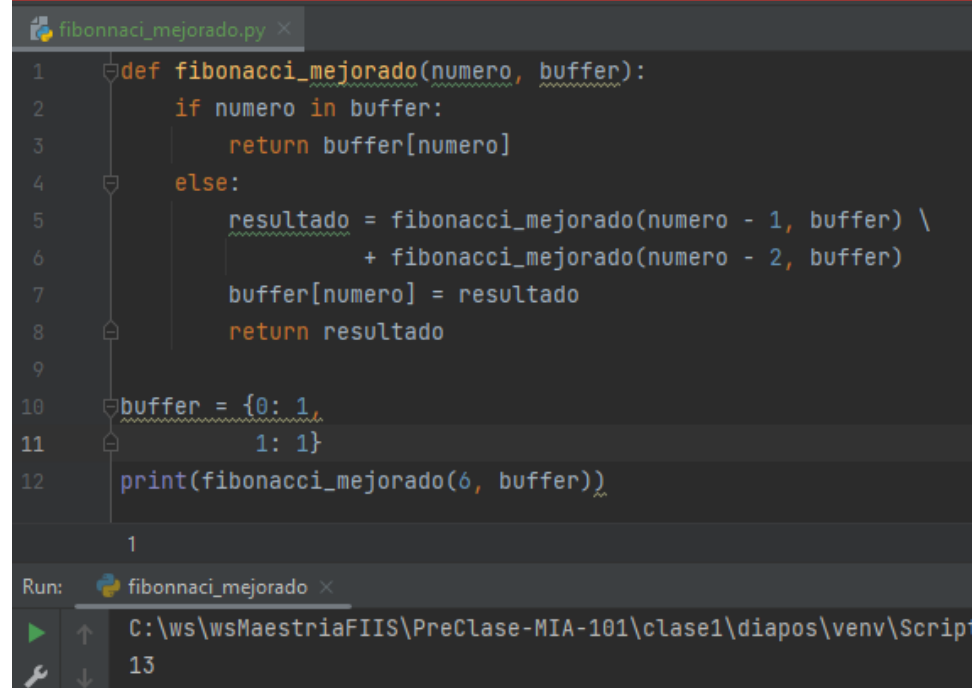



$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$



Fibonacci Mejorado

- Esta técnica es conocida como “memoización”.
- Primero, verificamos en `buffer` si hemos realizado el cálculo anteriormente.
- Durante la ejecución, se agregan nuevos valores al diccionario `buffer`.
- `fibonacci(34)` necesita 11,405,773 llamadas recurrentes. En contraste, `fibonacci_mejorado(34)` solo necesita 65.



```
fibonnaci_mejorado.py x
1 def fibonacci_mejorado(numero, buffer):
2     if numero in buffer:
3         return buffer[numero]
4     else:
5         resultado = fibonacci_mejorado(numero - 1, buffer) \
6                     + fibonacci_mejorado(numero - 2, buffer)
7         buffer[numero] = resultado
8         return resultado
9
10 buffer = {0: 1,
11           1: 1}
12 print(fibonacci_mejorado(6, buffer))

1
Run: fibonnaci_mejorado x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Script
13
```

AGENDA

1. INTRODUCCIÓN
2. ESTRUCTURAS DE CONTROL
3. CADENAS
4. FUNCIONES
5. TUPLAS, LISTAS, ALIASING, MUTABILIDAD Y CLONES
6. RECURSIVIDAD Y DICCIONARIOS
7. **EXCEPCIONES**



Excepciones

- Las excepciones se producen al producirse **condiciones inesperadas**.
- `ValueError` se produce cuando el operando tiene el tipo correcto, pero un valor ilegal.

```
exception1.py x
1 un_numero = int(input("Ingrese un numero:"))
2 otro_numero = int(input("Ingrese otro numero:"))
3 print("El conciente es: ", un_numero / otro_numero)
4
5 print("Bug en la entrada de usuario.")

Run: exception1 x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe C:/ws/wsMa
Ingrese un numero:cuatro
Traceback (most recent call last):
  File "C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\exceptions\exception1.py",
    un_numero = int(input("Ingrese un numero:"))
ValueError: invalid literal for int() with base 10: 'cuatro'

Process finished with exit code 1
```

Tipos de Excepciones



```
IPython: C:\Users\NESTOR
Type 'copyright', 'credits' or 'license' for more information
IPython 8.13.2 -- An enhanced Interactive Python. Type '?' for help.

In [1]: una_lista = [1,2,3]

In [2]: int(una_lista)
-----
TypeError                                Traceback (most recent call last)
Cell In[2], line 1
----> 1 int(una_lista)

TypeError: int() argument must be a string, a bytes-like object or a real number, not 'list'

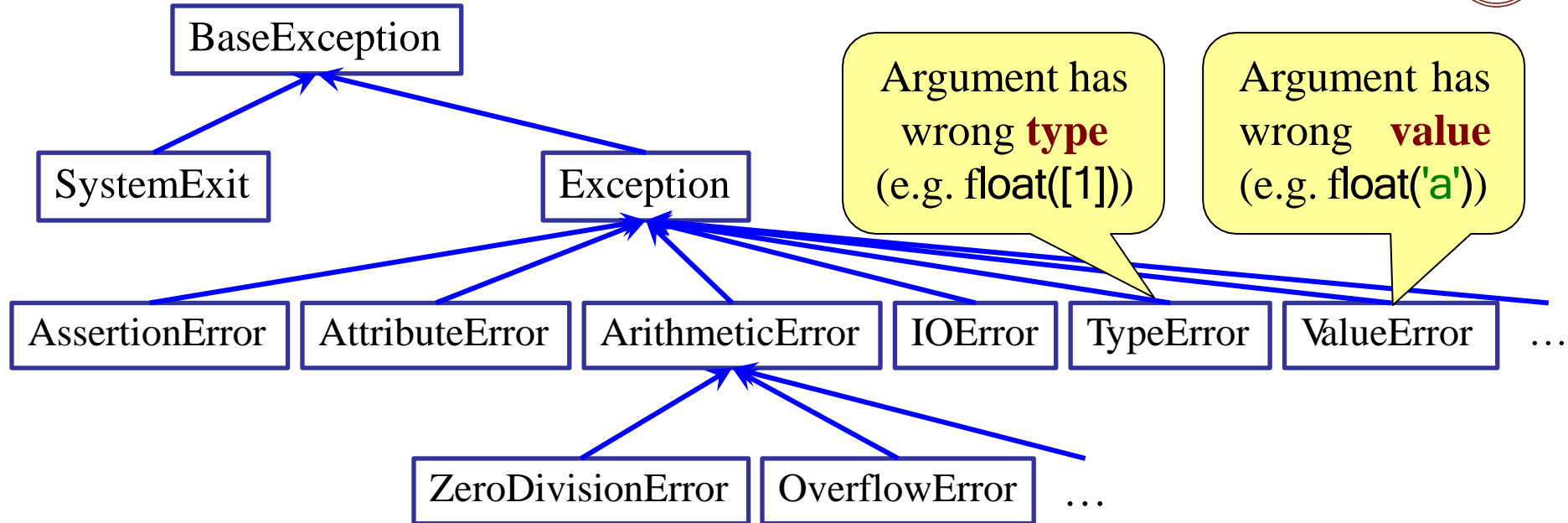
In [3]: print(otra_lista)
-----
NameError                                Traceback (most recent call last)
Cell In[3], line 1
----> 1 print(otra_lista)

NameError: name 'otra_lista' is not defined

In [4]: print("3"/4)
-----
TypeError                                Traceback (most recent call last)
Cell In[4], line 1
----> 1 print("3"/4)

TypeError: unsupported operand type(s) for /: 'str' and 'int'

In [5]: |
```

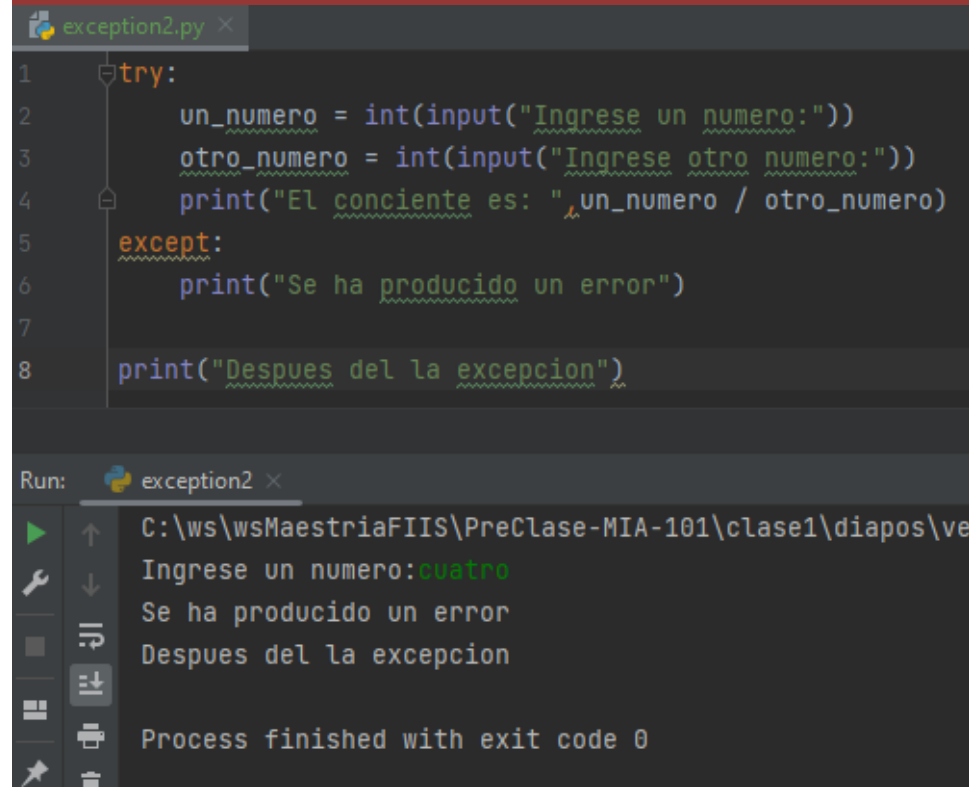


<http://docs.python.org/library/exceptions.html>

Why so many error types?

Manejo de Excepciones

- Podemos manejar las excepciones dentro del código Python.
- Cualquier excepción lanzada en el bloque `try` será manejada por el código en el bloque `except`.
- La ejecución del programa continúa luego de la ejecución del bloque `except`.



The screenshot displays a Python IDE with a file named `exception2.py`. The code defines a `try` block with three lines: `un_numero = int(input("Ingrese un numero:"))`, `otro_numero = int(input("Ingrese otro numero:"))`, and `print("El conciente es: ", un_numero / otro_numero)`. This is followed by an `except` block with one line: `print("Se ha producido un error")`. After the `except` block, there is a `print("Despues del la excepcion")` statement. Below the code editor, the 'Run' output window shows the execution results: the first input is 'cuatro', the second input is 'cuatro', the output is 'Se ha producido un error', and the program finishes with 'Exit code 0'.

```
1 try:
2     un_numero = int(input("Ingrese un numero:"))
3     otro_numero = int(input("Ingrese otro numero:"))
4     print("El conciente es: ", un_numero / otro_numero)
5 except:
6     print("Se ha producido un error")
7
8 print("Despues del la excepcion")
```

Run: exception2 x

C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\ve
Ingrese un numero:cuatro
Se ha producido un error
Despues del la excepcion
Process finished with exit code 0

Manejo de Excepciones

- Podemos definir bloques `except` por cada tipo de excepción.
- El último bloque `except` se ejecuta para excepciones que **no son** ni `ValueError` o `ZeroDivisionError`.

```
exception3.py x
1 try:
2     un_numero = int(input("Ingrese un numero:"))
3     otro_numero = int(input("Ingrese otro numero:"))
4     print("El conciante es: ", un_numero / otro_numero)
5
6 except ValueError:
7     print("No es posible convertir a un numero")
8 except ZeroDivisionError:
9     print("No es posible dividir entre cero")
10 except:
11     print("Error inesperado")
12
```

Run: exception3 x

```
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv
Ingrese un numero:10
Ingrese otro numero:0
No es posible dividir entre cero

Process finished with exit code 0
```


Lanzar Excepciones

- Mediante `raise`
`Exception("algo paso")`,
detenemos la ejecución y
hacemos visible el problema.
- Útil para indicar que el programa
no puede producir el resultado
esperado.

```
exception4.py
1 def dividir_listas(una_lista, otra_lista):
2     resultado = []
3     for indice in range(len(una_lista)):
4         try:
5             resultado.append(una_lista[indice] / otra_lista[indice])
6         except ZeroDivisionError:
7             resultado.append(float('nan')) # nan = Not a Number
8         except:
9             raise ValueError('Error en dividir las listas')
10
11     return resultado
12
13 primera_lista = [0, 1, 2]
14 segunda_lista = [3, 2, 1]
15
16 print(dividir_listas(segunda_lista), primera_lista)
17 print(dividir_listas(segunda_lista), primera_lista[1:])
```

Run: exception4

C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe C:/ws/wsMa
Traceback (most recent call last):
File "C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\exceptions\exception4.py",
 print(dividir_listas(segunda_lista), primera_lista)
TypeError: dividir_listas() missing 1 required positional argument: 'otra_lista'

Process finished with exit code 1

Promedio de Prácticas

- Dada una lista de listas, conteniendo nombres y notas, `incluir_promedio` genera una nueva lista que incluye el promedio de notas.
- La función falla en caso algún estudiante **no tenga notas**.

```
exception6.py
1 def incluir_promedio(nota_practicas):
2     notas_promedio = []
3     for alumno_notas in nota_practicas:
4         alumno = alumno_notas[0]
5         notas = alumno_notas[1]
6         notas_promedio.append([
7             alumno,
8             notas,
9             calcular_promedio(notas)
10        ])
11    return notas_promedio
12
13
14 def calcular_promedio(grades):
15     return sum(grades) / len(grades)
16
17
18 notas_algo = [['Paolo', 'Guerrero'], [10, 10, 10]],
19              [['Andres', 'Hurtado'], [10, 5, 12]],
20              [['Waldin', 'Saenz'], []]]
21
22 print(incluir_promedio(notas_algo))
23
Run: exception6
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe C:/ws/wsMa
Traceback (most recent call last):
  File "C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\exceptions\exception6.py", line 22, in <module>
    print(incluir_promedio(notas_algo))
  File "C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\exceptions\exception6.py", line 9, in incluir_promedio
    calcular_promedio(notas)
  File "C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\exceptions\exception6.py", line 15, in calcular_promedio
    return sum(grades) / len(grades)
ZeroDivisionError: division by zero
```

Promedio de Prácticas

- Podemos notificar al usuario del problema mediante un mensaje.
- Produce 0.0 en `calcular_promedio`, en caso de error.

Promedio de Prácticas

```
def incluir_promedio(nota_practicas):  
    notas_promedio = []  
    for alumno_notas in nota_practicas:  
        alumno = alumno_notas[0]  
        notas = alumno_notas[1]  
        notas_promedio.append([  
            alumno,  
            notas,  
            calcular_promedio(notas)]  
        )  
    return notas_promedio
```

```
def calcular_promedio(notas):  
    try:  
        return sum(notas) / len(notas)  
    except ZeroDivisionError:  
        print('Alerta: no existen notas')  
        return 0.0
```

```
notas_algo = [[['Paolo', 'Guerrero'], [10, 10, 10]],  
              [['Andres', 'Hurtado'], [10, 5, 12]],  
              [['Waldir', 'Saenz'], []]]
```

```
print(incluir_promedio(notas_algo))
```

✓ 0.0s

Alerta: no existen notas

```
[[['Paolo', 'Guerrero'], [10, 10, 10], 10.0], [['Andres', 'Hurtado'], [10, 5, 12], 9.0], [['Waldir', 'Saenz'], [], 0.0]]
```

Aserciones

- Se usan para verificar condiciones respecto al estado del programa.
- Mediante `assert`, lanzamos la excepción `AssertionError` en caso estas condiciones no se cumplan.
- Es una práctica de **programación defensiva**.
- Pueden usar para verificar **valores de entrada** en funciones.
- Pueden usarse también para verificar **el resultado de una función**, y evitar propagar errores.

```
aserciones.py x
1 def incluir_promedio(nota_practicas):
2     notas_promedio = []
3     for alumno_notas in nota_practicas:
4         alumno = alumno_notas[0]
5         notas = alumno_notas[1]
6         notas_promedio.append([
7             alumno,
8             notas,
9             calcular_promedio(notas)]
10        )
11    return notas_promedio
12
13
14 def calcular_promedio(notas):
15     assert len(notas) != 0, 'No existen notas'
16     return sum(notas) / len(notas)
17
18 notas_algo = [[['Paolo', 'Guerreiro'], [10, 10, 10]],
19               [['Andres', 'Hurtado'], [10, 5, 12]],
20               [['Waldir', 'Saenz'], []]]
21
22 print(incluir_promedio(notas_algo))
23
24 calcular_promedio()
```

Run: aserciones x

C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe C:/ws/wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe C:/ws/wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe C:/ws/wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\Scripts\python.exe

Traceback (most recent call last):

File "C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\exceptions\aserciones.py", line 16, in calcular_promedio(notas)

File "C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\exceptions\aserciones.py", line 15, in assert len(notas) != 0, 'No existen notas'

AssertionError: No existen notas



Libros

1. Ramalho, L. (2022). Fluent python. " O'Reilly Media, Inc.".
2. Guttag, J. V. (2021). Introduction to computation and programming using Python: with application to computational modeling and understanding data. Mit Press.

Cursos

1. "Introduction to Computer Science and Programming in Python" De MIT OpenCourseWare <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/>



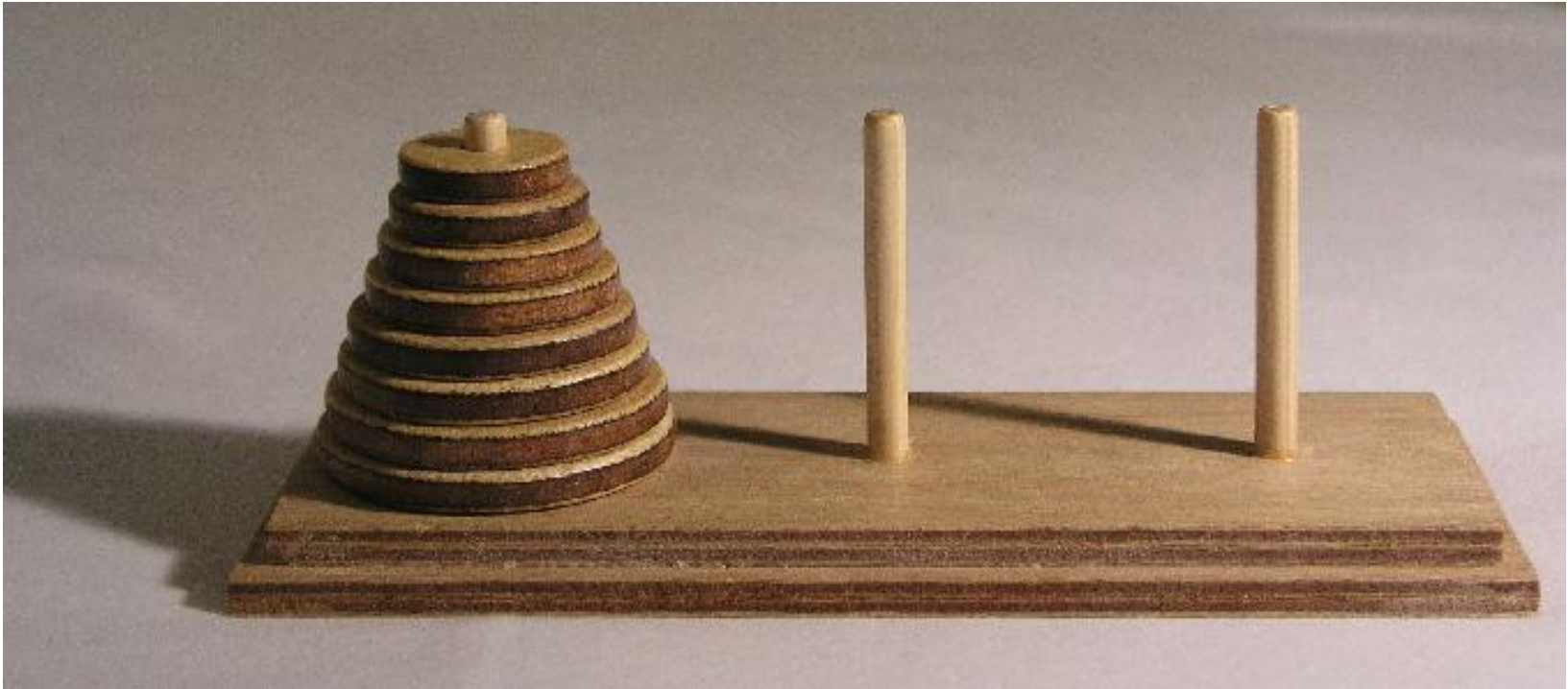
Gracias!!!







Torres de Hanoi





Torres de Hanoi

→ Usando recursividad:

- ◆ Primero, resolvemos un problema más pequeño.
- ◆ Luego, resolvemos el caso base.
- ◆ Finalmente, resolvemos otro problema más pequeño.

```
torres_hanoi.py x
4 def mover_discos(discos, origen, destino, libre):
5     if discos == 1:
6         mostrar_movimientos(origen, destino)
7     else:
8         mover_discos(discos - 1, origen, libre, destino)
9         mover_discos(1, origen, destino, libre)
10        mover_discos(discos - 1, libre, destino, origen)
11
12 print(mover_discos(5, "IZQ", "CENT", "DER"))
```

```
Run: torres_hanoi x
C:\ws\wsMaestriaFIIS\PreClase-MIA-101\clase1\diapos\venv\S
Mover discos desde IZQ hacia CENT
Mover discos desde IZQ hacia DER
Mover discos desde CENT hacia DER
Mover discos desde IZQ hacia CENT
Mover discos desde DER hacia IZQ
Mover discos desde DER hacia CENT
Mover discos desde IZQ hacia CENT
Mover discos desde IZQ hacia DER
Mover discos desde CENT hacia DER
Mover discos desde CENT hacia IZQ
Mover discos desde DER hacia IZQ
Mover discos desde CENT hacia DER
Mover discos desde IZQ hacia CENT
```