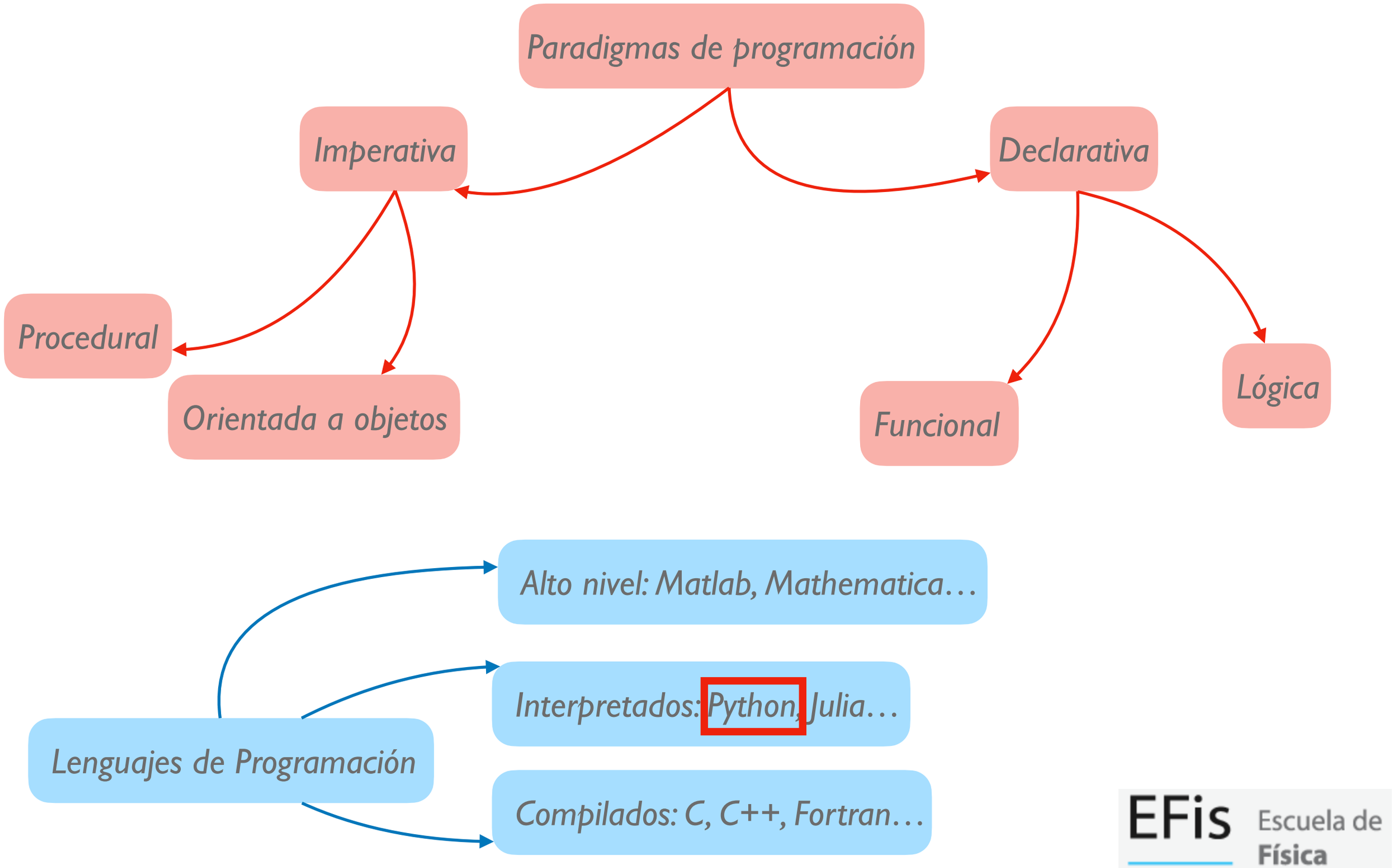
UNIVERSIDAD DE COSTA RICA

PROGRAMACIÓN INTERPRETADA: PYTHON

Prof. Marlon Brenes y Prof. Federico Muñoz
Escuela de Física, Universidad de Costa Rica

EFis Escuela de Física

Programación Intermedia: Guía de Curso



EFis Escuela de Física

Lenguajes interpretados

- Un programa llamado *interpretador* toma las instrucciones y las traduce a lenguaje de máquina

- No existe etapa de compilación

- A diferencia de un compilador, el cual toma todas las instrucciones y genera un ejecutable en lenguaje de máquina, un interpretador toma el código línea por línea y las ejecuta

- Razones para usar lenguajes interpretados:
  - Facilidad en la curva de aprendizaje
  - El número de bibliotecas es mucho más grande, en general
  - Excelente soporte para aplicaciones científicas
  - Ciclos de desarrollo más rápidos

Python: opciones de diseño

- Python es un lenguaje multi-paradigma:
  - Programación estructurada, OOP y funcional son posibles

- Ningún paradigma es forzado por el lenguaje (al contrario de, e.g., Java)

- Sintaxis usualmente limpia, fácil de entender pero puede llegar a ser difícil de optimizar

- Altamente extendible. La funcionalidad base es relativamente pequeña, mientras que las bibliotecas personalizadas son extensas

Hola mundo

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

print('Hello, world!')
```

- Ejecución:

```
(base) mbrenes@nia-login03:Semana_04$ ll
total 1
-rw-r----- 1 mbrenes dvira 48 Apr  4 00:20 hello.py
(base) mbrenes@nia-login03:Semana_04$ chmod +x hello.py
(base) mbrenes@nia-login03:Semana_04$ ./hello.py
Hello, world!
(base) mbrenes@nia-login03:Semana_04$
```

Esto se hace solo una vez!

Type system (sistema de interpretación)

```
1 # Strong Typing
2
3 'foo' + 5 # Esto es un error al runtime
```

```
1 # Dynamic Typing
2
3 a = 'foo'
4 b = 2 * a # b = 'foofoo'
5 a = 5
6 b = 2 * a # b = 10
```

```
1 # 'Duck' Typing
2
3 def foo(a, b):
4     return a + b
5
6 # Las funciones toman cualquier tipo
7 # como argumento. Si el tipo no calza,
8 # se devuelve un error al runtime
```

Sintaxis

- El espacio en blanco es significante! (`TAB`)
  - Los espacios denotan distintos scopes

```c
if(a > b){
  foo();
  bar();
}

baz();
```

C/C++

```python
if a > b:
    foo()
    bar()

baz()
```

Python

Flujos de control

```
1 for i in list_i:
2     baz(i)
```

```
1 while a > b:
2     foo()
3     bar()
```

```
1 if a > b:
2     foo()
3 elif b != c:
4     bar()
5 else:
6     baz()
```

```
1 pass
2
3 break
4
5 continue
```

Funciones

```python
def func(a, b, c):
    a = 3 * b
    return a + b -c
```

- Las funciones pueden ser pasadas como argumentos de funciones y las funciones pueden ser valores de retorno
  - En C/C++ esto requiere de bibliotecas externas (e.g., boost)

```python
def times_n(n):
    def helper(x):
        return n * x
    return helper

times_6 = times_n(6)
a = times_6(7) # a = 42
```

Manejo de excepciones

- Similar a la estructura `try/catch` en `C++`
  - La sintaxis en este caso es `try/except`

```
# Usualmente es mejor utilizar:
try:
    a = read_my_data()
except:
    print("Correupted data")

# que utilizar:
if consistent_data:
    a = read_my_data()
else:
    print("Corrupted data")
```

Expresiones

- Hay que tener cuidado con la división numérica
  - El resultado de la operación depende del tipo interpretado

```
1 # En Python 3:
2 a = 5 / 3 # a = 1.66666666667
3
4 a = 5 // 3 # 1
5
6 # Los booleanos y operaciones booleanas
7 # se escriben explícitamente
8 and or not
9 True False
```

Hileras de caracteres

- Los delimitadores de las hileras de caracteres son el operador ' y ", dependiendo de la necesidad
  - Secciones de código se pueden comentar encapsulando la sección con """ o '''

```
1 a = "Me dijo 'Hola, Marlon', ayer por la tarde"
2
3 # Comentarios de una sola línea con '#'
4 """
5 Más de una línea
6 de código
7 puede ser comentada
8 de esta manera
9 """
```

Formatos de impresión

- Existen dos formas de realizar distintos formatos de variables y caracteres
  - La primera es mediante el operador %

```
print("Hoy comí %d %s para el desayuno" % (2, "manzanas"))
```

  - La segunda es mediante el acceso al método `.format()`

```
print("Hoy comí {} {} para el desayuno".format(2, "manzanas"))
```

  - La segunda forma es más flexible

```
texto = "Hoy comí {num} {food}. Si, realmente comí {num}."
respuesta = texto.format(num = 12, food = "manzanas")

print(respuesta)
```

Arreglos y colecciones de datos

- En Python, no existe el concepto de pasar por valor/referencia/puntero
  - En lugar de esto, algunas estructuras son *mutables* y otras *inmutables*

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

# Esto es una lista y es *mutable*
# Es decir, si se pasa como argumento de una
# función, la función puede modificar su contenido
a = [3, 1, 'foo', 12.0]
def func(l):
    l[0] = 5
    return l
print(func(a)) # [5, 1, 'foo', 12.0]

# Esto es un tuple y es *inmutable*
b = (3, 1, 'foo', 12.0)
print(func(b)) # Runtime error. b es inmutable
```

- `mutable_vs_inmutable.py`

Arreglos y colecciones de datos

- Los diccionarios son mutables

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

# Los diccionarios son mutables
d = {'name': 'Marlon', 'age': 21}

print(d['name']) # Marlon
print(d['age']) # 21

def func(dd):
    dd['age'] = 20
    return dd

print(func(d))
```

- `dict.py`

Las listas se pueden recorrer de forma sencilla

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

pts = [
        (1, 3),
        (5, 6),
        (9, 10)
      ]

for i in pts:
    print(i)

for x, y in pts:
    print(x, 'and', y)
```

- `lists.py`

EFis Escuela de Física

Paquetes externos

- Python dispone de una cantidad increíble de paquetes externos para realizar distintas operaciones
  - numpy, scipy, matplotlib, Pandas, TensorFlow, PyTorch…
- Los paquetes se instalan mediante un *package manager*

Ejercicio de laboratorio

- Implementar el cálculo de pi en Python