

# Introducción a Python

**Christopher Flores Jara, M.Sc., D. Sc.**

Chillán, **10/07/2024**

# Contenidos

## 1. Breve descripción

## 2. Introducción a Python

Sintaxis, tipos de datos, operadores, estructuras de control, funciones

## 3. Introducción a la POO

Clases y objetos

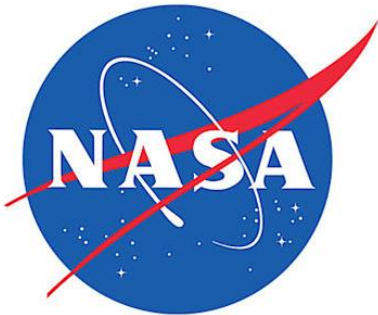
## 4. Procesamiento de datos

datos numéricos, gráficos


# Características

- ✓ Lenguaje interpretado
- ✓ Puede ser interactivo
- ✓ Fácil de aprender
- ✓ Desarrollo rápido de interfaces
- ✓ Open source
- ✓ Muchas bibliotecas disponibles
- ✓ Sintaxis permite un código legible
- ... etc

# ¿Quiénes usan Python?



# Lenguajes de programación más populares

Rank	Language	Type	Score
1	Python✓	  	100.0
2	Java✓	  	95.4
3	C✓	  	94.7
4	C++✓	  	92.4
5	JavaScript✓		88.1
6	C#✓	   	82.4
7	R✓		81.7

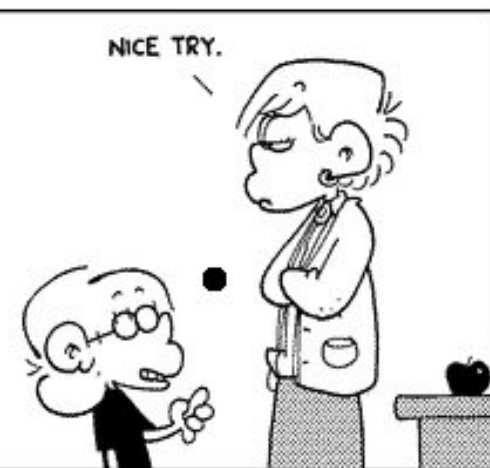
**Fuente:** IEEE Spectrum, 2021

# Let's go!

## C version

```
#include <stdio.h>
int main(void)
{
    int count;

    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.");
    return 0;
}
```



## Python version

```
print 'I will not throw paper airplanes in class.' * 500
```



# Sintaxis

## Identificadores

- Letras, underscore ( `_` ) y números
- No usar palabras reservadas
- Nombre de clases comienzan con mayúscula (convención)

## Líneas e indentación:

- Bloques se delimitan con indentación de líneas
- No se utilizan llaves `{}` ni punto y coma `;`
- Comentarios con `#`

# Operadores

- Numéricos: +, -, \*, /, \*\*, %, //
- Asignación: =, +=, -=, \*=, /=
- Comparación: ==, !=, <, >, <=, >=
- A nivel de bits: &, |, ^, ~, >>, <<
- Lógicos: and, or, not
- Membrecía: in, not in
- Identidad: is, is not



# Tipos de datos

- Números<sup>(\*)</sup>: int, float, long, complex, bool
- Secuencias: str<sup>(\*)</sup>, list, tuple<sup>(\*)</sup>, dict, set<sup>(\*)</sup>
- Clases
- Funciones

¡Todo tipo de datos es un objeto!

<sup>(\*)</sup> Inmutable

# Estructuras de control de flujo

## Condicionales:

- if, elif, else

## Ciclos:

- for, while, break, continue

## Funciones de python:

- zip, range, enumerate

# Algunas funciones estándares

Built-in Functions				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

# Actividades

- Verificar si un número ingresado por consola es par o impar
- Mostrar la cantidad de vocales de una palabra ingresada por consola. Utilizar un diccionario para almacenar la vocal y su frecuencia
- Obtener el promedio de una lista de números ingresados por consola

# Funciones

- Estructura general:

```
def nombre (parametros):  
    '''  
    documentacion de la funcion #opcional  
    '''  
  
    #cuerpo de la función  
    return expresión #opcional
```

- Llamada a función:

nombre(parametros)

# Actividades

- Crear una función que permita calcular la mediana de una lista de números:  
    mediana (lista)
- Crear una función que permita calcular el factorial de un número:  
    factorial (numero)
- Crear una función que permita invertir un número entero

# Clases y objetos

- Estructura general:

```
class Nombre (object):  
    """  
        documentacion de la clase #opcional  
    """  
  
    def __init__(self, parametros_clase):  
        self.atributo = parametro  
    def metodo (self, parametros):  
        #Cuerpo de la función  
        return expresión #opcional
```

- Creación de un objeto:

```
objeto = Nombre(parametros_clase)
```

# Actividades

- Crear una clase que permita obtener el valor aleatorio del lanzamiento de un dado. La clase `Dado(numero_caras)` debe contener:

Atributos: `numero_caras`, `valor`

Métodos: `__init__`, `lanzar`, `mostrar`

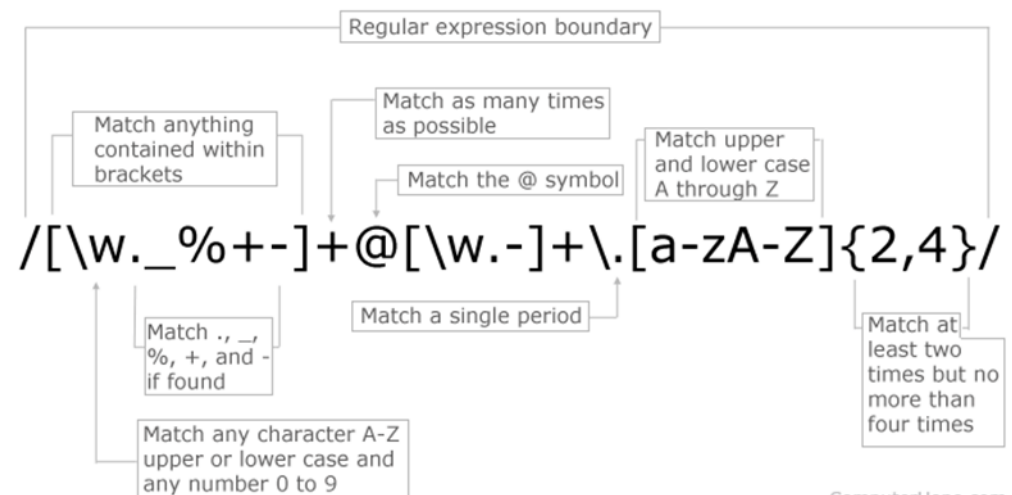
Sobrecargar el operador suma `__add__` para obtener la suma de 2 lanzamientos



# Expresiones regulares

- Secuencia de caracteres
- Patrones de texto
- En Python, import re
- Uso de metacaracteres (a continuación)
- Un ejemplo:

## Regular Expression E-mail Matching Example



# Expresiones regulares

## Metacaracteres

### Caracteres con significado especial

.	calza cualquier caracter menos \n
^	calza el principio del string
\\$	calza el final del string
*	calza 0 o más repeticiones de un caracter/regla
+	calza 1 o más repeticiones de un caracter/regla
?	calza 0 o 1 caracteres
*? , +? , ??	calces no voraces (i.e. <h1>title</h1>)
{m}	calza m veces la ER anterior
{m,n}	calza m a n veces (no voraz con ?)
\	escapa los metacaracteres (i.e. \*)
[]	conjunto de caracteres. Caracter – para rangos.
	A B calza A o B
()	calza la expresión entre parentesis completa (agrupa)

# Expresiones regulares

## Metacaracteres

### Caracteres con significado especial

<code>\number</code>	calza <i>number</i> veces lo que le precede
<code>\A</code>	calza <b>solo</b> al inicio del string
<code>\b</code>	calza el string vacio al principio/final de una palabra
<code>\B</code>	calza el string vacio no al principio/final de una palabra
<code>\d</code>	calza caracteres decimales ( <code>[0-9]</code> )
<code>\D</code>	calza todo caracter no decimal ( <code>[^0-9]</code> )
<code>\s</code>	calza todo caracter <i>blank</i> ( <code>[\t\n\r\f\v]</code> )
<code>\S</code>	calza todo caracter no <i>blank</i> ( <code>[^\t\n\r\f\v]</code> )
<code>\w</code>	calza todo caracter alfanumérico ( <code>[a-zA-Z0-9_]</code> )
<code>\W</code>	calza todo caracter no alfanumérico ( <code>[^a-zA-Z0-9_]</code> )

# Expresiones regulares

## Metacaracteres, algunos ejemplos:

```
import re
text = "tiene un imc      = 20.3, tiene obesidad incompatible"
#si, se escribe incompatible :)

find = re.findall(r"\d+", text)
print find
find = re.findall(r"\d+\.\d+", text)
print find
find = re.findall(r"imc", text)
print find
find = re.findall(r"\bimc\b", text)
print find
find = re.findall(r"^[^d]+", text)
print find
```

Salidas:

```
['20', '3']
['20.3']
['imc', 'imc']
['imc']
['tiene un imc      = ', '.', ', ', 'tiene obesidad incompatible']
```

# Expresiones regulares

## re.split(pattern, string)

- Similar a string.split("...")

```
import re

split = "A    1 B C".split(" ")
print split

split = re.split(r"\s", "A    1 B C")
print split

split = re.split(r"\s+", "A    1 B C")
print split

split = re.split(r"\s+", " A    1 B C ")
print split
print filter(None, split)
```

### Salidas:

```
['A', '', '', '', '1', '', 'B', 'C']
['A', '', '', '', '1', '', 'B', 'C']
['A', '1', 'B', 'C']
 ['', 'A', '1', 'B', 'C', '']
['A', '1', 'B', 'C']
```

# Expresiones regulares

## re.sub(pattern, replace, string)

- Similar a string.replace("...", "...")

```
import re
```

```
sub = "A    1  B C".replace(" ", "-")  
print sub
```

```
sub = re.sub(r"\s", "-", "A    1  B C")  
print sub
```

```
sub = re.sub(r"\s+", "-", "A    1  B C")  
print sub
```

```
text = "PhD Sheldon Cooper or Dr Sheldon Cooper"  
sub = re.sub(r"(PhD|Dr)\s+\w+\s+\w+", "**NOMBRE**", text)  
print sub
```

Salidas:

A----1--B-C

A----1--B-C

A-1-B-C

\*\*NOMBRE\*\* or \*\*NOMBRE\*\*

# Expresiones regulares

## re.findall(pattern, string)

- Entrega una lista de todas las ocurrencias

```
import re

text = "el paciente pesa 70 kg y tiene imc = 20"
find = re.findall(r"\d+", text)
if find:
    print "Encontrado"
    print find
else:
    print "No se encontro"
```

Salidas:

```
Encontrado
['70', '20']
```

# Expresiones regulares

## re.search()

- Permite capturar grupos utilizando ()

```
import re

text = "mi sitio web es www.udec.cl"
search = re.search(r"(\w+)\.(\w+)\.(cl|com)", text)
if search:
    print "Encontrado"
    print "Match:", search.group()
    print "Grupo 1:", search.group(1)
    print "Grupo 2:", search.group(2)
    print "Grupo 3:", search.group(3)
else:
    print "No se encontro"
```

### Salidas:

```
Encontrado
Match: www.udec.cl
Grupo 1: www
Grupo 2: udec
Grupo 3: cl
```



# Expresiones regulares

## Match en inicio y fin

- Uso de ^ para encontrar al inicio del string
- Uso de \$ para encontrar al fin del string
- Uso de ^ y \$ para una búsqueda exacta

```
import re
text = "9 alumnos en el curso de IATB."

find = re.findall(r"^d+", text)
print "Comienzo:", find

find = re.findall(r"[A-Z]+\.$", text)
print "Fin:", find

find = re.findall(r"^d+[\w\s]+\.$", text)
print "Exacto:", find
```

Salidas:

```
Comienzo: ['9']
Fin: ['IATB.']
Exacto: ['9 alumnos en el curso de IATB.']
```

# Expresiones regulares

## Flags

- Por eficiencia, es común compilar la expresión regular antes de utilizarla
- Se recomienda compilar si la regex se va a utilizar muchas veces en el código

Summary of Regex Flags		
syntax	long syntax	meaning
<code>re.I</code>	<code>re.IGNORECASE</code>	ignore case.
<code>re.M</code>	<code>re.MULTILINE</code>	make begin/end <code>{^, \$}</code> consider each line.
<code>re.S</code>	<code>re.DOTALL</code>	make <code>.</code> match newline too.
<code>re.U</code>	<code>re.UNICODE</code>	make <code>{\w, \W, \b, \B}</code> follow Unicode rules.
<code>re.L</code>	<code>re.LOCALE</code>	make <code>{\w, \W, \b, \B}</code> follow locale.
<code>re.X</code>	<code>re.VERBOSE</code>	allow comment in regex.

# Expresiones regulares

## Flags

```
import re
text = "Curso IATB-2018"
find = re.findall(r"[a-z]+\-\d+", text)
print find
find = re.findall(r"[a-z]+\-\d+", text, re.I)
print find
```

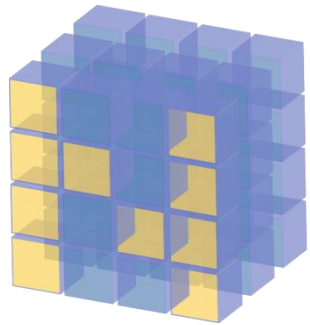
```
text = u"La Universidad de Concepción está en Concepción"
find = re.findall(r"[A-Z]\w+", text)
print find
find = re.findall(r"[A-Z]\w+", text, re.U)
print find
for f in find:
    print f.encode("utf8")
```

Salidas:

```
[]
['IATB-2018']
```

```
[u'La', u'Universidad', u'Concepci', u'Concepci']
[u'La', u'Universidad', u'Concepci\xef3n', u'Concepci\xef3n']
La
Universidad
Concepción
Concepción
```

# Procesamiento de datos numéricos y gráficos



NumPy

Pandas



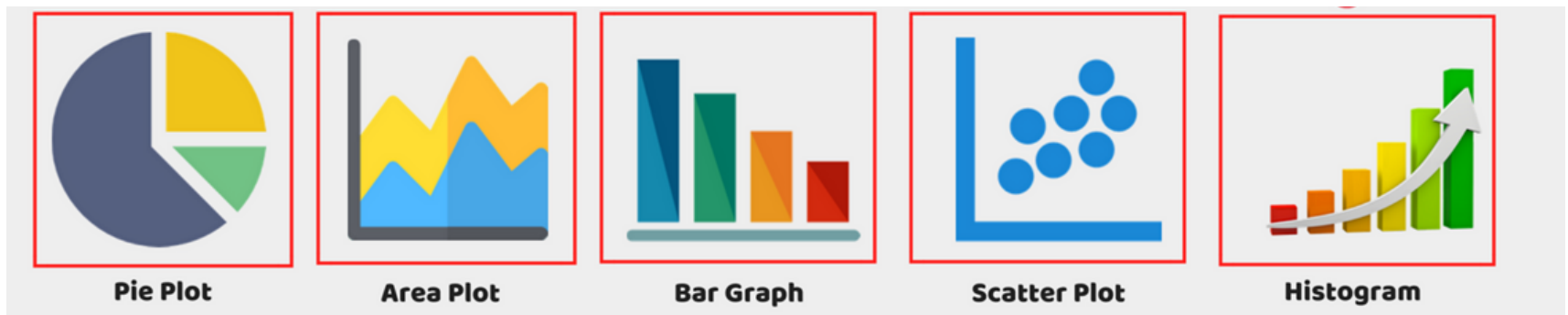
**matplotlib**

# Procesamiento de datos numéricos y gráficos

- Arreglos (ejemplo 4 filas, 4 columnas):

(fila 0, col 0)		...	(fila 0, col 3)
...		...	
		...	
(fila 3, col 0)		...	(fila 3, col 3)

- Tipos de gráfico



# Actividades

- Cargar los datos de masa y altura: Mostrar un gráfico de dispersión de ambas variables.  
¿Cuántos pacientes tienen obesidad ( $\text{imc} \geq 30$ )?
- Cargar una imagen en RGB. Convertirla a escala de grises y binaria. Mostrar todas los 3 tipos de imágenes en un subplot.  
$$\text{grises} = 0.2889 * R + 0.5870 * G + 0.1140 * B$$
$$\text{binaria} = (\text{grises} > \text{umbral}) * 255$$