# Programación para Física y Astronomía

## Departamento de Física.

Profesores Claudia Loyola / Alejandro Llanquihuen / Joaquín Peralta

Primer Semestre / 2022

Universidad Andrés Bello

Universidad Andrés Bello®

python powered

# Resumen

# Funciones, Paquetes y Módulos

# Paquetes

Existen muchas operaciones que se pueden desarrollar en un programa, las cuales son mucho más complicadas que la aritmética sencilla.

1. Multiplicar matrices.
2. Calcular logaritmo.
3. Hacer gráficos, entre otras.

Python posee una gran variedad de **funciones** que permiten hacer ciertas tareas. Estas **funciones** son divididas en paquetes y cada paquete posee un nombre con el que podemos referenciarle.

**Paquetes**

Colección de funciones útiles relacionadas.

# Paquetes

Antes de usar cualquiera de estas funciones, hay que importar la función desde el paquete principal.

**Example (`log` del paque `math`)**

```
from math import log
x=log(2.5)
```

**algunas funciones del paquete `math`**

| | |
|---|---|
| log | logaritmo natural |
| log10 | logaritmo base 10 |
| exp | exponencial |
| sin, cos, tan | seno, coseno, tangente (argumento en radianes) |
| asin, acos, atan | seno, coseno, tangente (argumento en radianes) |
| sinh, cosh, tanh | seno, coseno, tangente hiperbólico |
| sqrt | raíz cuadrada |

# Paquetes

Si tenemos dudas respecto de las funciones disponibles en un paquete, el comando *dir(paquete)* nos lista todas funciones pertenecientes a un paquete.

**Example**

```
>>>import math
>>>dir(math)
['__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',
'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow',
'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
'trunc']
```

# Paquetes

Algo más acerca del paquete *math*

- *math* también contiene funciones menos comunes como la *función de error de Gauss* y la *función gamma*.
- Además objetos que no son funciones como $e$ y $\pi$. Notar que *pi* no necesita paréntesis debido a que **no es** una función.

```
1 from math import pi
2 print(pi**2)
```

Otra ventaja es que se pueden importar varias funciones en la misma línea:

**Example**

```
from math import log, exp
```

# Paquetes

O incluso, importar **todas** las funciones de un paquete:

**Example**

```
from math import *
```

Sin embargo, la práctica de importar todas las funciones de un paquete nos puede causar inconvenientes. Por ejemplo, si casualmente hemos nombrado alguna de nuestras variables o funciones usando un nombre contenido en el paquete, puede haber conflictos o ejecuciones erradas en el código.

# Módulos

Algunos paquetes que son muy grandes, son convenientemente divididos en pequeños subpaquetes denominados **módulos**.
Los **módulos** son referenciados como
*nombre_paquete.nombre_modulo*.

**Example (importando del módulo `linalg` del paquete `numpy`)**

```
from numpy.linalg import inv
```

# Poniendo en práctica

Supongamos que la posición de un punto en el espacio está dada en coordenadas polares $(r, \theta)$, y nosotros necesitamos convertir éstas al espacio de coordenadas cartesianas. ¿Cómo se escribe un programa que realice dicha operación?

1. Solicitar al usuario los valores de $r$ y $\theta$.
2. Convertir estos valores al espacio cartesiano usando las fórmulas:

$$x = rcos\theta, y = rsin\theta$$

3. Imprimir el resultado.

# Poniendo en práctica

Supongamos que la posición de un punto en el espacio está dada en coordenadas polares $(r, \theta)$, y nosotros necesitamos convertir éstas al espacio de coordenadas cartesianas. ¿Cómo se escribe un programa que realice dicha operación?

1. Solicitar al usuario los valores de $r$ y $\theta$.
2. Convertir estos valores al espacio cartesiano usando las fórmulas:

$$x = rcos\theta, y = rsin\theta$$

3. Imprimir el resultado.

```
1  from math import sin,cos,pi
2  r = float(input("Ingrese r: "))
3  d = float(input("Ingrese theta en grados: "))
4  theta = d*pi/180
5  x = r*cos(theta)
6  y = r*sin(theta)
7  print(f'x = {x}, y = {y}')
```

# Funciones: A tener en cuenta

1. **Funciones integradas**
   Existe un pequeño conjunto de funciones que están integradas en Python y no provienen de ningún paquete. Estas se denominan **funciones integradas** y lo son por ejemplo: `float`, `int`, `complex`, `abs`, `input` y `print`.
2. **Declaración de comentario**
   En Python cualquier línea de que comience con un *hash* "`#`" es ignorado completamente por el interprete.

**Example**

```
# Hola, esta linea no hace nada!
```

Estas líneas son denominadas comentarios y son muy usadas por los programadores para documentar sus programas en relación con lo que hace el código en general y algunas partes en específico. Un buen código siempre está documentado!.

# Estructuras de control

# Estructuras de control

Los programas revisados hasta el momento han sido lineales, en el sentido que van de una instrucción a la siguiente, desde el comienzo hasta el final del código o programa. Para cambiar este comportamiento lineal y hacer más versátil la programación, los lenguajes de programación incorporan múltiples **estructuras de control** que permiten saltar de una sección de código a otra, omitir una sección de código o repetir muchas veces cierta sección de código.

# La declaración “IF”

Usamos la declaración **IF** cuando queremos que nuestro programa haga algo sólo si cierta condición es satisfecha primero.

```
1 x = int(input("Ingrese un entero no mayor a diez:"))
2 if x>10:
3   print("Usted ingreso un numero mayor que diez.")
4   print("Permitame corregirle.")
5   x = 10
6 print("Su número es",x)
```

Este programa utiliza el condicional **IF** para corregir el número ingresado, en caso de ser un valor mayor a diez.

**Nota**:

Fijarse muy bien en el bloque de instrucciones pertenecientes al condicional **IF**. Este bloque se encuentra sangrado (*indented*).

# La declaración “IF”

Existen diferentes tipos de condiciones que pueden ser usadas en la declaración IF.

**title**

| | |
|---|---|
| `if x==1:` | Comprueba si $x = 1$. Notar el signo igual doble. |
| `if x>1:` | Comprueba si $x > 1$ |
| `if x>=1:` | Comprueba si $x \geq 1$ |
| `if x<1:` | Comprueba si $x < 1$ |
| `if x<=1:` | Comprueba si $x \leq 1$ |
| `if x!=1:` | Comprueba si $x \neq 1$ |

Se pueden combinar dos o más condiciones en una simple declaración usando el “*or*” y el “*and*”.

```
1 if x>10 or x<1:
2   #realizar algo
3 if x<=10 and x>=1:
4   #realizar otra cosa
```

# La declaración “IF”

Existen dos declaraciones complementarias del **IF** que son muy útiles: *else* y *elif*.

```
if x>10:
  print("Su numero es mayor que diez.")
else:
  print("Su numero esta correcto. Nada por hacer.")
```

```
if x>10:
  print("Su numero es mayor que diez.")
elif x>9:
  print("Su numero esta bien, pero bastante cercano.")
else:
  print("Su numero esta correcto. Nada por hacer.")
```

# La declaración “WHILE”

La declaración **WHILE** es usada comúnmente cuando necesitamos asegurar que se cumpla alguna condición en un programa o para continuar realizando una operación hasta que se alcanza cierto punto o situación.

**Pensemos** en el ejemplo de **IF**: Pero esta vez solicitaremos que ingrese nuevamente un número hasta que esté correcto.

## Algo así

```
Ingrese un número entero no mayor a diez: 11
El número es mayor que diez. Por favor trate nuevamente.
Ingrese un número entero no mayor a diez: 57
El número es mayor que diez. Por favor trate nuevamente.
Ingrese un número entero no mayor a diez: 100
El número es mayor que diez. Por favor trate nuevamente.
Ingrese un número entero no mayor a diez: 5
Su número es 5
```

# La declaración “WHILE”

El siguiente código permite obtener el resultado anterior. La declaración **WHILE** (*mientras*) indica que mientras la variable ***x*** sea mayor que diez el bloque sangrado a continuación del **WHILE** se repetirá todas la veces que sea necesario.

```
1 x = int(input("Ingrese numero entero no mayor a diez: "))
2 while x>10:
3   print("El numero es mayor que diez. Reintente.")
4   x = int(input("Ingrese numero entero no mayor a diez: "))
5 print("Su numero es",x)
```

Se pueden especificar dos o más criterios usando “*and*” y “*or*”.

```
1 x = int(input("Ingrese un numero positivo no mayor a diez: "))
2 while x>10 or x<1:
3   print("El numero es negativo o es mayor que diez. Reintente.")
4   x = int(input("Ingrese un numero positivo no mayor a diez: "))
5 print("Su numero es",x)
```

# La secuencia de Fibonacci

# La secuencia de Fibonacci

Los números de Fibonacci son una secuencia de enteros en los cuales cada uno de ellos es la suma de los dos números previos, teniendo en cuenta que los primeros dos números son 1 y 1. De este modo, los primeros números de la secuencia son: 1, 1, 2, 3, 5, 8, 13, 21... Relación de concurrencia de la secuencia de Fibonacci:

$$F_n = F_{n-1} + F_{n-2}$$

con valores semilla o iniciales:

$$F_1 = 1, F_2 = 1$$

# La secuencia de Fibonacci

Proponga un programa que calcule los números de Fibonacci hasta el 1000. Considere que el programa necesita mantener un registro de los dos números más recientes de la secuencia y entonces sumarlos para calcular el siguiente.

```
1  f1 = 1
2  f2 = 1
3  next = f1 + f2
4  while f1<=1000:
5    print(f1)
6    f1 = f2
7    f2 = next
8    next = f1 + f2
```

# Funciones definidas por el usuario

# Funciones definidas por el usuario

Existen muchas situaciones en Física Computacional donde se necesitan funciones especializadas para desarrollar un cálculo y Python nos permite definir nuestras propias funciones.
Supongamos que necesitamos calcular el factorial de enteros:

$$n! = \prod_{k=1}^{n} k$$

El siguiente código nos permite calcular el factorial de un número.

```
1 n=int(input("Ingrese un valor entero positivo:"))
2 f=1
3 k=1
4 while(k<=n):
5     f*= k
6     k+=1
7 print(n,"! = ",f)
```

¿Pero qué pasa si necesito calcular varias veces el factorial en

# Funciones definidas por el usuario

La solución más conveniente es definir nuestra propia función factorial.

```
1  #Aqui comienza mi funcion
2  def factorial(n):
3   f=1
4   k=1
5   while(k<=n):
6      f*= k
7      k+=1
8   return f
9
10 #Aqui comienza mi progama principal
11 n=int(input("Ingrese un valor entero positivo:"))
12 print(n,"! = ",factorial(n))
```

La instrucción **def** nos permite definir una función en nuestro programa. El bloque de código sangrado a partir de la instrucción **def** pertenece a la definición de dicha función.

# Funciones definidas por el usuario

**Funcionamiento**

Cuando se escribe `factorial(n)` dentro del programa principal, la función `factorial()` es *llamada* con el argumento `n` y la ejecución del programa continúa en la definición de la función. Al llegar al final del bloque, la instrucción `return` indica que la función regresa o retorna un valor al programa principal, donde continua la ejecución.

# Recursividad

# Recursividad

**Recursividad** en ciencia computacional es un método donde la solución a un problema depende de las soluciones a pequeñas instancias del mismo problema.
Los lenguajes de programación soportan recursividad por medio de la *autollamada* a funciones dentro del código o programa.

```
1  def factorial(n):
2    if n==1:
3      return 1
4    else:
5      return n*factorial(n-1)
```

# Actividades

# Actividades para hoy

- Realice las actividades de la guía.

# Fin