

# Estructuras de los programas

## Temario de la clase

- Funciones
- Variables locales y globales
- Programas principales o scripts.
- Librerías.
- Aproximación de funciones.

**Fecha Primer Parcial: 20 de Setiembre! (2 semanas)**

El parcial es hasta Guía 6: Vectores y Matrices

Que sucede si queremos realizar el mismo conjunto de instrucciones en numerosas partes de un programa?

Tenemos que repetir estas instrucciones en todas partes? (usamos el copy paste?)

Una de las herramientas mas útiles de python son las **funciones**, que actuan de una forma muy similar a las funciones matemáticas, las funciones tienen un conjunto de **variables de entrada**, y tienen una o varias **variables de salida**/resultados.

## Funciones o rutinas

La **función/ subrutina/ rutina** es una estructura de programación:

- ▶ Las funciones estan definidas por un **nombre**.
- ▶ Las funciones internamente tienen **un conjunto de instrucciones**.
- ▶ En general van a tener **variables de entrada** o **argumentos de entrada**, cuando se llama a la función se le pasa los valores de estas variables.
- ▶ La función va a procesar/utilizar estas variables de entrada en el conjunto de instrucciones internas.
- ▶ Finalmente, cuando se termina la función se puede dar **variables de salida** o **argumentos de salida**. Estos son los resultados de la rutina que se pasan al programa principal.

## Sintaxis de la función en python

Las funciones se definen con un **def** luego el nombre de la función y entre paréntesis los argumentos de entrada:

```
def funcio_3Dgral(x,y,z):  
    Instrucciones  
    ...  
    return v1,v2
```

después de los **argumentos** (entre paréntesis) ponemos los dos puntos.

Como en los condicionales (if) y los loops (for).

Luego viene el cuerpo de la función. **Se debe tabular a todo el cuerpo de la función.**

El cuerpo de la función es el conjunto de instrucciones que queremos realizar. Termina con un `return` y las variables de salida.

Si no hay variables de salida (rara vez), no es necesario el `return`.

## Comentario de las funciones

Se aconseja después de la línea def, un comentario explicando que es lo que hace la función, las variables de entrada y las de salida.

```
def funcion_3Dgral(x, y, z, vx, vy, vz) :  
    '''  
    Calcula posicion siguiente de una misil balistico  
    dada la posicion actual y la velocidad  
    variables de entrada  
    x,y,z posicion actual  
    vx,vy,vz velocidad  
    variables de salida  
    xnew,ynew,znew posicion nueva  
    @ Manuel [2019-09-19]  
    @ TODO Agregar calculo de velocidad siguiente  
    '''
```

## Ejemplo: sintaxis de la función en python.

Todas las instrucciones de la función van tabuladas a 4 espacios. Y cuando queremos que el flujo vuelva al programa principal con el resultado de la función usamos `return`,

```
def suma (a,b) :  
    '''  
    Calcula la suma de dos variables de entrada  
    '''  
    resultado=a + b  
    return resultado  
def otrafuncion(x,y,z) :  
    resto
```

Cuando termina la función después del `return` se vuelve la tabulación al original.

## Ejemplo 2: Transformación de la temperatura

Supongamos que en nuestro programa tenemos que hacer cambios de grados Celsius a Fahrenheit, sabemos que la función es  $F(C) = \frac{9}{5}C + 32$ ,

```
def trans_c2f(tcel):  
    " Transforma temperatura de Celsius a Fahrenheit"  
    tfah=9./5.*tcel+32.0  
    return tfah
```

**Variable de entrada:** tcel temperatura en Celsius. **Variable de salida:** tfah temperatura en Fahrenheit.

## Uso o “llamada” de la función.

Luego en el programa principal llamamos a la función:

```
ta = 10
F1 = trans_c2f(ta)
temp = trans_c2f(15.5)
print ( trans_c2f(ta+1) )
sum_temp = trans_c2f(10.0) + trans_c2f(20.0)
```

- ▶ Notar todas las veces que estoy llamando a la función!
- ▶ Aun cuando la función se utilice una sola vez en el programa principal, conceptualmente un programa es mucho mas claro cuando cada “función” esta definida en una “función independiente”.
- ▶ Aun cuando la función sea de una sola línea también es conveniente.



## Regla: Funciones todo funciones

REGLA DE ORO: programa principales pequeños que llaman a funciones.

Es mucho mas fácil programar con funciones:

- ▶ Las funciones son unidades básicas independientes, esto nos permite reciclarlas, ej. en cualquier programa que necesite transformar temperatura puedo utilizar la función `trans_c2f`.
- ▶ Además tiene muy bien definido todo lo que necesita para funcionar (Argumentos de entrada) y cuales son los resultados (variables de salidas).
- ▶ El debugging de una función de pocas líneas es mucho mas sencillo (**Se aísla del resto**).
- ▶ Las variables que se utilizan en las funciones son locales. Veamos...

## Variables locales. Nacen y mueren en la función

Las variables que se definan en una función son **variables locales**, es decir se crean para la función, pero luego en el return se destruyen y no afectan el resto del programa.

```
ta = 10
def trans_c2f(tcel):
    factor=9./5.
    ta=factor*tcel+32.0
    return ta
F1 = trans_c2f(10)
print 'Variable ta: ',ta
print 'Variable factor: ',factor
```

Que da el print de ta? el valor que se calcula en la función o el que esta en el programa principal?

Que da el print del factor?

## Variables locales. Nacen y mueren en la función

**Rta 1:** 10 (el valor del programa principal, la función no cambia el valor de ta).

**Rta 2:** La variable factor no existe en el programa principal, si esta en la función NO puede ser vista por el programa principal (se destruye a la salida).

Aun cuando se use una **variable del mismo nombre** en la función que estamos usando en el programa principal los valores no alteran los del programa principal (son lugares de memoria distintos).

## Variables globales

Las variables que se definan fuera de la función pueden ser utilizadas en la función:

```
factor = 9./5.  
def trans_c2f(tcel):  
    print 'Factor adentro de la funcion: ',factor  
    ta=factor*tcel+32.0  
    return ta  
F1 = trans_c2f(ta)  
print 'Variable ta: ',ta  
print 'Variable factor: ',factor
```

Que da el print de ta?

Que da el print del factor?

## Cambio de valor en variables globales?

¿Las variables que se definan fuera de la función pueden ser utilizadas en la función?

```
factor = 5.  
def trans_c2f(tcel):  
    factor = 9./5.  
    ta=factor*tcel+32.0  
    return ta  
F1 = trans_c2f(20)  
print 'Variable ta: ',ta  
print 'Variable factor: ',factor
```

¿Qué da el print del factor?

¿Pueden describir cual es la diferencia con el caso anterior? ¿Cual es la lógica?

## Cambio de valor en variables globales

Para redefinir una variable adentro de la función se utiliza:

```
factor = 5.  
def trans_c2f(tcel):  
    global factor # aqui avisamos que estamos usando  
                  # la variable global "factor"  
    factor = 9./5. # aqui NO esta creando una variable local  
                  # pero redefiniendo la variable global  
    ta=factor*tcel+32.0  
    return ta  
F1 = trans_c2f(ta)  
print 'Variable ta: ',ta  
print 'Variable factor: ',factor
```

Que da el print del factor?

Pueden describir cual es la diferencia con el caso anterior?

## Argumentos opcionales. Valores por default

Existen **argumentos opcionales** que puedo definir o no en el llamado de la función.

```
def posrel(t, v0, g=9.81)
    y=v0*t- 0.5*g*t**2
    return y
```

A esta función la puedo llamar como:

```
y=posrel(10, 5)
```

para todos los casos que este en la Tierra.

Si quiero calcular la posición en Marte

```
y=posrel(10, 5, g=3.711)
```

Si llama a la función sin ese argumento tomará el valor por default. Si la llamo con el argumento puedo ponerle el valor que quiero.

Notar que como es opcional le pongo el nombre del argumento en la

## Switches como argumentos

En el problema de caída libre queremos tener una opción para que imprima el resultado en la función:

```
def posrel(t, v0, g=9.81, salida=None):  
    y=v0*t- 0.5*g*t**2  
    if salida is not None:  
        print 'La posicion es: ',y  
    return y
```

Si la variable salida esta definida que imprima el resultado (esto actua de switch, pero tambien podría tener valores).

Esta función se puede llamar como:

```
posrel(10, 5, salida=1)  
posrel(10, 5, g=3.711)  
posrel(10, 5, g=3.711, salida=1)
```



## Donde ubicamos a las funciones?

Si estamos escribiendo en un solo archivo al programa principal y las funciones:

Las funciones deben ir antes del llamado a éstas.

```
# Primero Funcion
def trans_c2f(tcel) :
    factor=9./5.
    ta=factor*tcel+32.0
    return ta
# Llamando a la funcion desde el programa principal
F1 = trans_c2f(10.)
```

Esto es así porque python es un interprete, entonces necesita tener “interpretada” la función, antes que se la quiera utilizar (de lo contrario no sabría lo que es “trans\_c2f”).

## Donde ubicamos a las funciones?. Librerías

Las funciones pueden ir en un archivo aparte en el mismo directorio: una **librería**

También pueden ir en un subdirectorio pero en este caso necesito agregar un archivo vacío:

```
__init__.py
```

Esto le avisa a python que es un directorio python y eventualmente se puede utilizar este archivo para inicializar librerías del subdirectorio.

Para utilizar una librería nuestra es equivalente a las de python. Le tenemos que decir que la importe:

```
import libreria
```

Si es en un subdirectorio:

```
import dir.libreria
```

## Donde ubicamos a las funciones?. Importación de librerías

Supongamos que ubicamos en el archivo `transformaciones.py` las funciones `trans_c2f` y `trans_f2c`. En el programa principal `transforma.py` hacemos:

```
#!/usr/bin/env python
"Transforma de grados centigrados a fahrenheit y viceversa"
import transformaciones as tr

print 'Que desea calcular: '
opt=input(' (1) Celsius a Fahrenheit, (2) Fahrenheit a Celsius:')
if opt == 1:
    tcel=input('Ingrese Temperatura en Celsius')
    tfah=tr.trans_c2f(tcel)
    print 'Temperatura de ',tcel,' Celsius es ',tfah,' Fahrenheit'
elif opt == 2:
    tfah=input('Ingrese temperatura en Fahrenheit')
    tcel=tr.trans_f2c(tfah)
    print 'Temperatura de ',tfah,' Fahrenheit es ',tcel,' Celsius'
```

Por convención es al comienzo!.

## Librerías de python

- ▶ `math` Funciones matemáticas
- ▶ `sys` Parametros del sistema
- ▶ `os` Variables y comandos de la shell.
- ▶ `numpy` Matrices y vectores. Operaciones.
- ▶ `numpy.random` Generador de números aleatorios.
- ▶ `matplotlib` Graficación.
- ▶ `tkinter`. Interface a Tcl/Tk Interfaces gráficas para int. usuario.
- ▶ `threading` Paralelismo por hilos.
- ▶ `multiprocessing` Paralelismo por procesos.
- ▶ `scipy`, `sklearn`, `tensorflow`, `keras`, etc aprendizaje automatizado

## Formas de uso

Si se importa la libreria en forma directa:

```
import math
a=math.log(15.)
```

Si se importa la libreria pero redefiniendola:

```
import math as ma
a=ma.log(15.)
```

Si importamos a la función:

```
from math import log
a=log(15.)
```

Si queremos importar todas las funciones de una libreria (no recomendada):

```
from math import *
a=log(15.)
```

## Nuestras propias librerías en carpetas python

Las funciones pueden ir en un archivo en otro directorio especial.

Supongamos que tenemos muchas funciones en archivos y entonces hacemos una carpeta, “funciones” y ponemos todos los archivos “.py” allí (Excepto el programa principal).

Esta es la forma en la cual están escritas las librerías “numpy”, “os” etc.

Para que el “funciones” funcione como un directorio python, debemos poner un archivo adentro `__init__.py` (solo contiene un espacio “ ” por el momento o vacío)

Poner el archivo `transformaciones.py` en la carpeta funciones

```
$ mv transformaciones.py funciones
```

Luego en el programa principal `transforma.py` lo único que hacemos es:

```
import funciones.transformaciones as tr
```

El resto igual.

## Aproximación de funciones

¿Como hacemos para determinar la función  $\exp(-x^2 - 1)$  en  $x = 1.5$  con una computadora que solo sabe sumar y multiplicar?

Existen metodos para aproximar funciones a traves de polinomios:

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n = \sum_{i=0}^n a_ix^i$$

Si lo que queremos es evaluar la funcion alrededor de algun punto lo que podemos hacer es un desarrollo de Taylor.

## Series de Taylor

Supongamos el desarrollo alrededor de  $x_0$  de la función  $f(x)$

$$f(x) \approx f(x_0) + \frac{df}{dx}(x_0)(x - x_0) + \frac{1}{2} \frac{d^2f}{dx^2}(x_0)(x - x_0)^2 + \cdots + \frac{1}{n!} \frac{d^n f}{dx^n}(x_0)(x - x_0)^n$$

$$f(x) \approx \sum_{i=0}^n \frac{1}{i!} \frac{d^i f}{dx^i}(x_0)(x - x_0)^i$$

La idea es que si  $x - x_0 \ll 1$ , entonces los términos de la serie disminuyen con la potencia y esto permite que el error de la aproximación este acotado (los primeros terminos de la serie son los dominantes).



## Aproximación de funciones

$$f(x) = \sum_{i=0}^n \frac{1}{i!} \frac{d^i f}{dx^i}(x_0)(x - x_0)^i + \epsilon^{n+1}$$

La serie es convergente (para funciones infinitamente diferenciables).

El error de un orden  $n$  lo podemos calcular con el orden siguiente ya que las contribuciones de los términos sucesivos disminuyen  $(x - x_0)^n < (x - x_0)^{n+1}$ ,

$$\epsilon^{n+1} \approx \frac{d^{n+1}f}{dx^{n+1}}(x_0)(x - x_0)^{n+1}$$

Esta metodología solo es válida cuando  $x$  esta muy cercano a  $x_0$ .

Aproximación alrededor de un punto.

Cuando queremos aproximar a una función en un **intervalo** vamos a usar la **interpolación polinómica**.