Cronograma

- Solución de algunos ejercicios de la clase pasada
- Funciones
  - Conceptos generales
  - Definición e invocación
  - Parámetros por defecto
  - Documentación
  - Variables locales
  - Cadena de llamadas
  - Pasaje de parámetros
  - Inmutabilidad
  - Devolviendo resultados

Posibles soluciones

```
STORED_PASS= "pepe"
ATTEMPTS = 3
i = 1
user_pass = input("ingrese su clave: ")
while user_pass != STORED_PASS and i < ATTEMPTS:
    print ("clave erronea")
    user_pass = input("ingrese su clave: ")
    i += 1
if (i == ATTEMPTS):
        print ("No adivinaste la clave")
else:
        print ("Adivino la clave")
```

Posibles soluciones (2)

```
i = 0
match = False
while (not match) and (i < ATTEMPTS):
    user_pass = input("ingrese su clave: ")
    if user_pass == STORED_PASS:
        print ("clave correcta")
        match = True
    else:
        print ("clave erronea")
        i += 1
        time.sleep(i)
```

Posibles soluciones (3)

```
import random

stored_number = random.randrange(1, 11)
match = False
while not match:
    user_number = int(input("ingrese su numero entre 1 y 10: "))
    if user_number == stored_number:
        print ("acierto")
        match = True
    else:
        if user_number > stored_number:
            print ("no acierto {} es mayor" . format (user_number))
        else:
            print ("no acierto {} es menor" . format(user_number))
```

Funciones

- Ya hemos visto algunas llamadas a funciones:
  - type(“33”)
    - <type ’string’>
  - len(“palabra”)
    - 7
- El valor o variable, llamado el **argumento ó parámetro** de la función, ha de estar encerrado entre paréntesis,y si hay más de uno, el separador que debe usarse es la coma.

Funciones (2) - Definición

Añadiendo funciones nuevas:

- Podemos agregar nuevas funciones definidas por nosotros.
- La creación de nuevas funciones para resolver problemas particulares es algo muy útil en todos los lenguajes de programación.
- La sintaxis para la definición de una función es:

```
def  NOMBRE([LISTA DE PARAMETROS]):
    SENTENCIAS
```

Funciones (3) - Invocación

- Definición e invocación de una función

Parámetros

```
def potencia(base,exponente):
        return base ** exponente
```

Argumentos

```
print potencia(2,4)
```

Funciones (4)

- Algunas de las funciones que vimos necesitan de uno o varios argumentos.

```
def imprimeDoble(paso):
        print paso, paso
```

¿Pero de qué tipo van a ser los argumentos?

No lo sabemos hasta ejecutar la función

La función ***imprimeDoble*** sirve con cualquier tipo (de dato) que se pueda imprimir

```
imprimeDoble('Queso')
Queso Queso
imprimeDoble(5)
5 5
imprimeDoble(3.14159)
3.14159 3.14159
```

Funciones (5)

Parámetros por defecto:

- Podemos asignar valores por defecto a nuestros parámetros.
- En el caso de que no se indique ningún valor para ese parámetro se utiliza el valor por defecto.

```
def imprimir(texto, veces = 1):
        print( veces * texto)
```

Funciones (6)

```
def funcion (a1=1, a2=2, a3=3):
    print (a1, a2, a3)

funcion()
funcion("a", "b", "c")
funcion("a", "b")
funcion("a")
funcion(a3="hola")

# salidas
(1, 2, 3)
('a', 'b', 'c')
('a', 'b', 3)
('a', 2, 3)
(1, 2, 'hola')
```

Funciones (7) - Documentación

- Cada función debería realizar una tarea especifica.
- Cuando tenemos una gran cantidad de funciones definidas se nos puede hacer difícil saber exactamente que hace cada función.
- Es extremadamente importante documentar nuestras funciones.
- La documentación de una función se coloca luego del encabezado de la función, en un párrafo encerrado entre triple comillas

Funciones (8)

Por ejemplo

```
def saludo(nombre):
        """ Imprime por pantalla un saludo, dirigido a la persona que se indica por parámetro. """
        print("Hola {}".format(nombre),"!")
        print("Estoy programando en Python")
```

- Cuando documentamos nuestras funciones podemos acceder a su documentación a través de la función *help(saludo)*

Funciones (9)

Variables locales

- Cada función define un nuevo namespace
- Todos los parámetros y variables utilizadas dentro de una función son variables locales. Por lo que solo existen y viven mientras dura la función
- Obtenemos un error al intentar acceder a la variable a fuera de la función.

```
from math import sqrt

def area_triangulo(a,b,c)
 s = (a+b+c)/2.0
 return sqrt (s*(s-a)*(s-b)*(s-c))

print area_triangulo(1,3,2.5)
print a
```

Devolver múltiples resultados

- Una función puede devolver un resultado utilizando la palabra reservada ***return***
- Pero, ¿cómo hacemos para devolver más de un resultado?
- Podemos retornar una **tupla** de valores.
  - return (n1,n2,n3…..Nn)

Devolver múltiples resultados (2)

```
def nombres_3_sobrinos ():
        """ Funcion que retorna el nombre de los tres sobrinos del Pato Donald """
        pato1 = "Hugo"
        pato2 = "Paco"
        pato3 = "Luis"
        return (pato1,pato2,pato3)

print nombres_3_sobrinos()

(a,b,c) = nombres_3_sobrinos()
```

Recursividad

Ya vimos que una función puede llamar a otra.

- También podemos hacer que una función se llame a si misma.

```
1 def cuenta_atras(n):
2         if n == 0:
3                 print "Despegando!"
4         else:
5                 print n
6                 cuenta_atras(n-1)
7
8 cuenta_atras(n-1)
```

Recursividad (2)

- **Ejemplos clásicos:**

  Factorial de un número

```
def factorial(n):
        if n == 0:
                return 1
        else:
                return n * factorial(n-1)
```

  Función Fibonacci

```
def fibonacci (n):
        if n == 0 or n == 1:
                return 1
        else:
                return fibonacci(n-1) + fibonacci(n-2)
```

Recursividad (3)

- Infinita: Si una recursión no alcanza nunca el caso base, seguirá haciendo llamadas recursivas para siempre y nunca terminará.

```
def funcion():
        funcion()
```

Python informara de un mensaje de error cuando se alcance el nivel máximo de recursividad

RuntimeError: maximum recursion depth exceeded

Ejercicios

Ejercicio 1

- Escribir una función que reciba un número n por parámetro e imprima los primeros n números triangulares, junto con su índice.
- Los números triangulares se obtienen mediante la suma de los números naturales desde 1 hasta n. Es decir, si se piden los primeros 5 números triangulares, el programa debe imprimir:
  - 1 - 1
  - 2 - 3
  - 3 - 6
  - 4 - 10
  - 5 - 15

Ejercicios (2)

- **Ejercicio 2**

  Escribe una función que muestre por pantalla los números múltiplos de 7 entre el 1 y n, donde n es un parámetro

  - Utiliza range(1, n+1) en un bucle for con los if necesarios.
  - Después haz lo mismo empleando un range con tres parámetros.

Ejercicios (3)

- **Ejercicio** 3

  Previo: explicar parámetro end de la función print

  Escriba un programa que pida anchura y altura de un rectángulo y llame a una función para dibujarlo. Se debe dibujar con caracteres ' * '

  Anchura del rectángulo: 5

  Altura del rectángulo: 3

  * * * * *

  * * * * *

  * * * * *

Ejercicios (4)

- **Ejercicio 4**

  Ídem a ejercicio 3, pero que reciba un parámetro más: el carácter a imprimir (por defecto: '*')

- **Ejercicio 5**

  Escribir una función que reciba un entero n y represente un triángulo isósceles con sus dos lados iguales de largo n.