IWI 131- PROGRAMACIÓN

Ayudantía 5

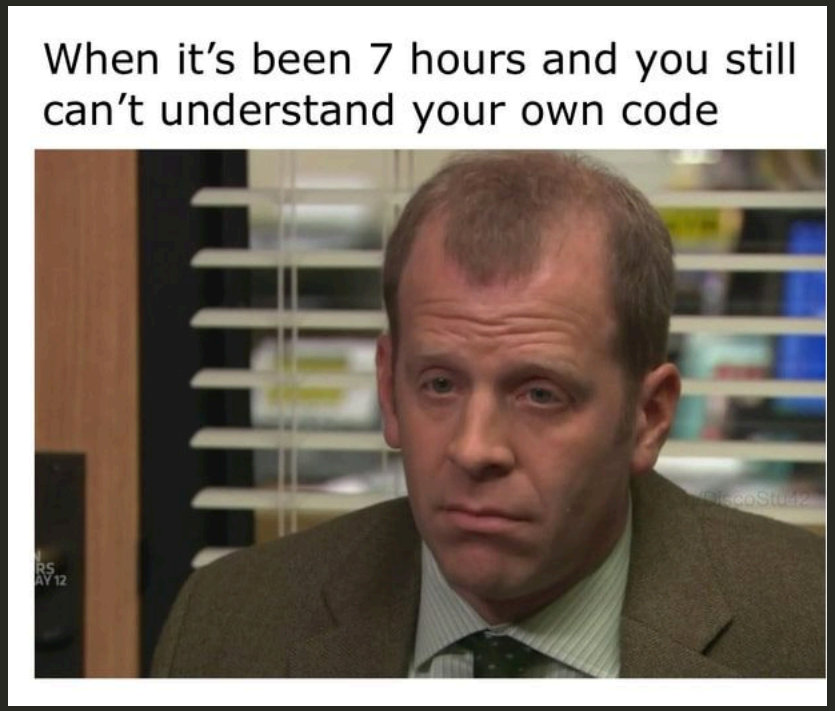
Ayudante: Anastasiia Fedorova

Paralelo: 212

Fecha: 6.05.2020

¿FUNCIONES?

Cuando nuestros programas crecen en tamaño, tenemos que hacer algo para facilitar el desarrollo y la revisión de errores. Además, siempre es mejor generalizar el código. Por se devidie el código en subprogramas, los cuales los denominamos funciones.



¿FUNCIONES?

El propósito de funciones es escribir el código solo una vez y poder llamarlo de cualquier lugar en nuestro programa, tantas veces como sea necesario, usando solamente 1 linea – la llamada a la función.

```
def mostrarCantidadNaipes(self, screen, listaTextos):
    screen.blit(listaTextos[0],(self.u2.getX() + 33, self.u2.get
    list(map(lambda i: screen.blit(listaTextos[i + 1], (self.bot
    ) + 33, self.bot_ai[i].getY() + 107)), [i for i in range(5)]
'''
mostrarOponentes(self, screen)
|   Función que muestra los jackets de oponentes del usuario.
|   CONCEPTOS DE CURSO: Funciones de orden superior. Formas func
'''
def mostrarOponentes(self, screen):
    list(map(lambda i: screen.blit(self.bot_ai[i].getImg(),
        (self.bot_ai[i].getX(), self.bot_ai[i].getY())), [i for

'''
gameStart(self, screen)
|   Función responsable de inciar el juego. Se crea la baraja, s
|   las cartas, se genera la mano visible del usuario y la lista
'''
def gameStart(self, screen):
    self.jugadores = self.crearJugadores(self.nJugadores)
    self.baraja = Baraja()
    # Crea la baraja para la partida
    self.repartirCartas(self.jugadores, screen)
    self.trump = self.makeTrump()
    self.makeFirstPlayer()
    self.manoVisible = self.jugadores[0].manoAcotada(self.listpo
```

Motivación – reutilizar el código ya hecho en vez de repetirlo (un ejemplo – un juego de cartas)

LLAMADAS, DEFINICIÓN, RETURN

Las instrucciones que ejecuta la función las llamamos definición de la función.

Las instrucciones se ejecutan a través de la llamada a la función.

Una vez que termine la ejecución de la función, esta retorna un valor específico (depende de lo que hace la función) en el lugar donde fue ejecutada la llamada.

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


x =  int(input("x= "))
print(x,"!=", factorial(x))
```

DEFINICIÓN

La definición de la función consiste en escribir, primero, la definición de la cabecera de la función – como se llama y que es lo que recibe:

```
def calcular_distancia(x, y):
```

El segundo paso es definir el cuerpo de la función – que es lo que esta hará. Y en el último paso, se define que es lo que la función retorna.

```
def calcular_distancia(x, y):
    dist = sqrt(x** 2 + y ** 2)
    return dist
```

LLAMADAS

Una vez definida la función, la podemos llamar desde cualquier lugar de nuestro código. Para ejecutar la llamada, simplemente tenemos que poner el nombre de la función y rellenar los argumentos.

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


n = int(input("Cuantos factoriales quiere calcular? "))
while n:
    num = int(input("Ingrese un numero: "))
    fact = factorial(num)
    if fact != -1:
        print("Su factorial es:", fact)
    else:
        print("No es posible calcular")
    n -= 1
```

PARAMETROS

Muchas de las funciones reciben argumentos (no todas) – elementos cuyo valor se copia a las variables que están en los paréntesis de la llamada a la función.

La función recibe argumentos en las variables, llamadas parametros – son variables especiales, que pertenecen solamente a la función y existen solo cuando esta se está ejecutando.

```
def calcular_distancia(x, y):
    dist = sqrt(x** 2 + y ** 2)
    return dist
```

Variable x local
10

Variable y local
5

```
dist = calcular_distancia(10, 5):
```

Se copia el valor de argumentos a los parámetros.

VARIABLES GLOBALES Y LOCALES

Sabemos que las funciones son unas subrutinas. Las variables que existen dentro de la función no son accesibles a las demás funciones o al programa principal, aunque tengan el mismo nombre.

```
def subrutina():
    a = 7
    print(a)
    return


subrutina()
print(a)
```

¿Por qué?

```
7
Traceback (most recent call last):
  File "t.py", line 7, in <module>
    print(a)
NameError: name 'a' is not defined
```

VARIABLES GLOBALES Y LOCALES

Sabemos que las funciones son unas subrutinas. Las variables que existen dentro de la función no son accesibles a las demás funciones o al programa principal, aunque tengan el mismo nombre.

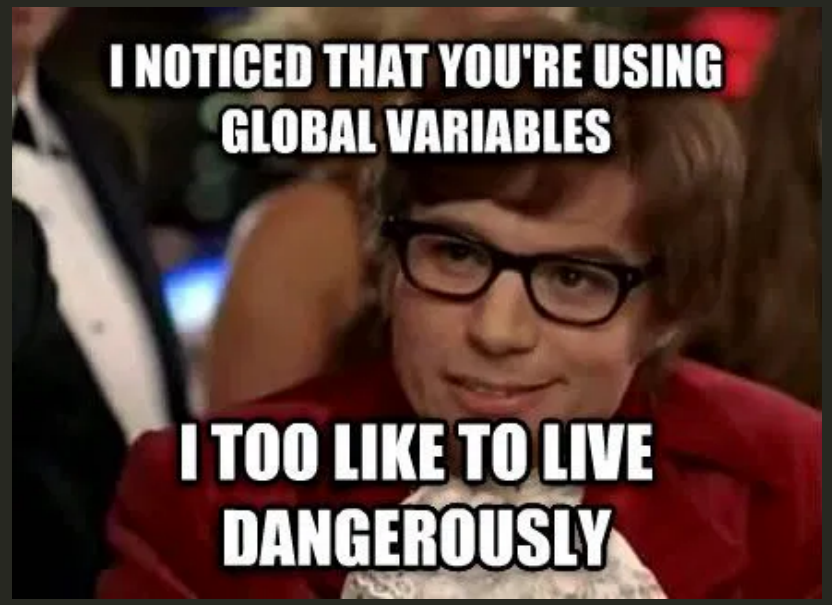
```
def subrutina():
    a = 7
    print(a)
    return


a = 9
subrutina()
print(a)
```

a =9 no es la misma a = 7

```
7
9
```

VARIABLES GLOBALES Y LOCALES

Las variables globales son accesibles a todo nivel de código. Se aconseja no utilizar las variables globales en las funciones – estas deben recibir como parámetros todo lo que necesiten para funcionar bien.




```
def subrutina():
    print(a)
    return


a = 5
subrutina()
print(a)
```

Funciona, imprime

5

5

Pero es mejor no hacer esto.

PRINT Y RETURN

Como ya vimos, la sentencia return no es lo mismo que la sentencia print. La primera nos permite retornar un valor calculado y guardarlo en una variable para poder procesarlo después. La segunda, en cambio, solamente imprime la información por pantalla – no es posible captarla ni trabajarla.

```
def sum_1(x):
    return x + 1


x = 10
print(x + 1)
print(x)
x = sum_1()
print(x)
```

```
11
10
11
```

Como se puede ver, con print() el valor de x no cambia, aunque hagamos x + 1. Sin embargo, si guardamos el return de la funcion sum_1 en x, su valor sí que cambia.

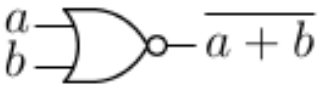
EJERCICIO 1:

Su ayudante está cursando INF 245: Arquirectura y Organización de Computadores, en el cual se trabaja mucho con la logica booleana y compuertas lógicas.

Se les pide crear funciones, que simulan el comportamiento de las operaciones NOR, XNOR, NAND. Estas deben retornar un valor booleano como resultado de la operación.

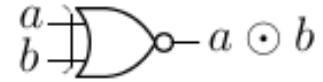
NOR

a	b	NOR
0	0	1
0	1	0
1	0	0
1	1	0



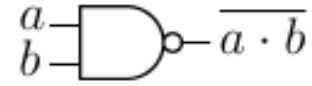
XNOR

a	b	XNOR
0	0	1
0	1	0
1	0	0
1	1	1



NAND

a	b	NAND
0	0	1
0	1	1
1	0	1
1	1	0



EJERCICIO 1: SOLUCIÓN

```python
def nor(a, b):
    return not(a or b)


def nand(a, b):
    return not(a and b)


def xnor(a, b):
    return a == b
```

C1 2018-1 (Nota: el pdf tiene `print w, d` sin paréntesis porque en 2018 usabamos Python 2.7)

EJERCICIO 2: RUTEO

```
def foo(h):
    if h % 2 != 0:
        return False
    return True
def too(y):
    if foo(y):
        y = y / 2
    else:
        y = y + 3
    return y

t = True
w = 18
d = 4
while(t):
    w = too(w)
    d = too(d)
    if w - 1 == d:
        t = False
print(w, d)
```

Determine que es lo que hace la función too(y)

Global			foo	too
t	w	d	h	y

EJERCICIO 2: SOLUCIÓN

```
def foo(h):
    if h % 2 != 0:
        return False
    return True
def too(y):
    if foo(y):
        y = y / 2
    else:
        y = y + 3
    return y

t = True
w = 18
d = 4
while(t):
    w = too(w)
    d = too(d)
    if w - 1 == d:
        t = False
print(w, d)
```

Pantalla:

3 2

Si el número es par, retorna su mitad, si impar, retorna este número, aumentado en 3 unidades.

Global			foo	too
t	w	d	h	y
True				
	18			
		4		
				18
			18	
				9
	9			
				4
			4	
				2
		2		
				9
			9	
				12
	12			
				2
			2	
				1
		1		
				12
			12	
				6
	6			
				1
			1	
				4
		4		
				6
			6	
				3
	3			
				4
			4	
				2
		2		
False				

EJERCICIO 3: ECUACIÓN DE PRIMER GRADO

Escriba un programa que pida los coeficientes de una ecuación de primer grado: `ax + b = 0`,y que entregue la solución.

Una ecuación de primer grado puede: tener solución única, tener infinitas soluciones, o no tener soluciones.

Input:
Ingrese a: **0**
Ingrese b: **3**

Output:
Sin solución

Input:
Ingrese a: **4**
Ingrese b: **2**

Output:
Solución única: -0.5

Input:
Ingrese a: **0**
Ingrese b: **0**

Output:
No hay solución única.

EJERCICIO 3: SOLUCIÓN

```
def solucion(a, b):
    if a == 0:
        if b == 0:
            print("No hay solucion unica")
            return
        print("No hay solucion")
    else:
        print("Solucion unica:", -b / a)


a = int(input("Ingrese coef. a: "))
b = int(input("Ingrese coef. b: "))
solucion(a, b)
```

EJERCICIO 4:

Lea tres números enteros y imprima el máximo y el mínimo de los tres. Para ello, escriba funciones que retornan máximo y mínimo entre 2 números.

Input:
7
21
-14

Output:
-14
21

EJERCICIO 4: SOLUCIÓN

```
def mmin(n, m):
    if n >= m:
        return m
    return n

def mmax(n, m):
    if m >= n:
        return m
    return n
```

```
#imprime min y max al ejecutar
#se les ocurre como ahorarse un llamado a
la funcion mmin?
def order3(a, b, c):
    print(mmin(mmin(a, b), mmin(b, c)))
    print(mmax(mmax(a, b), mmax(b,c)))

a = int(input("Ingrese a:"))
b = int(input("Ingrese b:"))
c = int(input("Ingrese c:"))
order3(a, b, c)
```

EJERCICIO 4: SOLUCIÓN*

```
def mmin(n, m):
    if n >= m:
        return m
    return n

def mmax(n, m):
    if m >= n:
        return m
    return n
```

```
#imprime min y max al ejecutar
def order3(a, b, c):
    print(mmin(mmin(a, b), c))
    print(mmax(mmax(a, b), c))

a = int(input("Ingrese a:"))
b = int(input("Ingrese b:"))
c = int(input("Ingrese c:"))
order3(a, b, c)
```

EJERCICIO 5: CATA DE VINOS

Un sansano que tomó el Taller de Cata de Vinos con su profesor se encuentra en una bodega. En cierto momento se le presentan varias copas para catar el vino. Se sabe que de estas copas 6 contienen Cabernet y 4– Merlot.

Si se sabe que la probabilidad de que n vasos que elija el sansano sean de mismo tipo es:

$$\frac{\frac{6!}{(6-n)!\,n!}+\frac{4!}{(4-n)!\,n!}}{\frac{10!}{(10-n)!\,n!}}$$

Ejemplo:
Si n = 2, la probabilidad es de 0.47 o 47%

Escribir el programa que pueda calcular la probabilidad para un 2<= n <= 4. Reutilice la función de calcular factorial.

EJERCICIO 5: SOLUCIÓN

```
from random import randint


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

```
def combi(n, k):
    return factorial(n) / (factorial(n - k) * factorial(k))


n = randint(2, 4)
cabernet = combi(6, n)
merlot = combi(4, n)
total = combi(10, n)
print("La probabilidad de elegir", n, "vaso(s) de mismo vino es",
      round((cabernet + merlot) / total, 2))
```