UNIVERSIDAD NACIONAL
GUILLERMO BROWN

CLASE 2 - Unidad I

Funciones y Recursividad

ESTRUCTURAS DE DATOS (271)

Clase N. 2. Unidad I.

Clase 2: AGENDA

AGENDA

- **Temario:**
  - Revisión de Conceptos: funciones
  - Conceptos de Recursividad
  - Introducción TDA
- **Ejemplos** en Lenguajes **Python**
- **Temas relacionados y links de interés**
- **Práctica**
- **Consultas**
- **Cierre de la clase**

Clase 2: TEMARIO. Funciones

Declaración de una Función:

Para definir una función usamos la palabra reservada def, luego escribimos la **signatura** de la función, esto es:

- el **nombre** de dicha función y si existen
- los **parámetros** que recibe.

Como siempre, debemos termina la línea con ‘: ’ para indicar que a continuación habrá un bloque de código. Obviamente, luego respetar la indentación.

```
def nombreFunción(arg_1 , ... , arg_r) :
        {código de la función}
        ...
        {más código de la función}

#programa principal
Código del programa
```

Al igual que cualquier bloque de código indentado, debe contener al menos una linea, en otro caso deberemos escribir **pass** luego de los ‘: ’ .

Clase 2: TEMARIO. Funciones

unab

Parámetros por referencia:

- Cuando se pasa una variable a una función como parámetro por referencia, los cambios que se efectúan sobre dicha variable dentro de la función se mantienen, incluso después de que haya finalizado la función.
- Los cambios producidos en parámetros por referencia son permanentes, ya que no se pasa a la función el valor que contiene la variable, sino la dirección de memoria de la variable.

Parámetros por valor:

- Cuando se pasa un parámetro por valor a una función se guarda en memoria una copia temporal de la variable, dentro de la función solo se utiliza la copia, la original nunca se toca.

Clase 2: TEMARIO. Funciones

unab

Declaración de una Función:

```
1   #funciones
2
3   def perimetro_rectangulo(la,an):
4       per=(la*2)+(an*2)
5       return (per)
6
7   def peri_octa(l):
8       peri=l*8
9       return (peri)
10
11  #Programa principal
12
13  largo=float (input('Ingrese el largo del rectángulo: '))
14  ancho=float (input('Ingrese el ancho del rectángulo: '))
15  print ('El perímetro del rectángulo es: ', perimetro_rectangulo(largo,ancho))
16  print()
17  print(perimetro_rectangulo(largo,ancho))
18  0c=float (input('Ingrese el largo del octágono regular: '))
19  print ('El perímetro del octágono regular es: ', peri_octa(0c))
20
```

Clase 2: TEMARIO. Recursividad

unab

Definición:

La recursividad o recursión es un concepto que proviene de las matemáticas, y que aplicado al mundo de la programación nos permite resolver problemas o tareas donde las mismas pueden ser divididas en subtareas cuya funcionalidad es la misma. Dado que los subproblemas a resolver son de la misma naturaleza, se puede usar la misma función para resolverlos.

Dicho de otra manera, una función recursiva es aquella que **está definida en función de sí misma**, por lo que se llama repetidamente a sí misma hasta llegar a un punto de salida.

Cualquier función recursiva tiene **dos secciones** de código claramente divididas:

- Por un lado, tenemos la sección en la que la función se llama a sí misma.
- Por otro lado, **tiene que existir siempre una condición** en la que la función **retorna sin volver a llamarse**.

  (Es muy importante porque de lo contrario, la función se llamaría de manera indefinida)

Clase 2: TEMARIO. Recursividad

Ejemplo: El factorial de un número entero no negativo **n** es el producto de todos los enteros desde **n** hasta 1.

5! = 1*2*3*4*5 5!= 5*4*3*2*1 5!= 120

- Formalmente: $n! = \prod_{k=1}^{k=n} k$

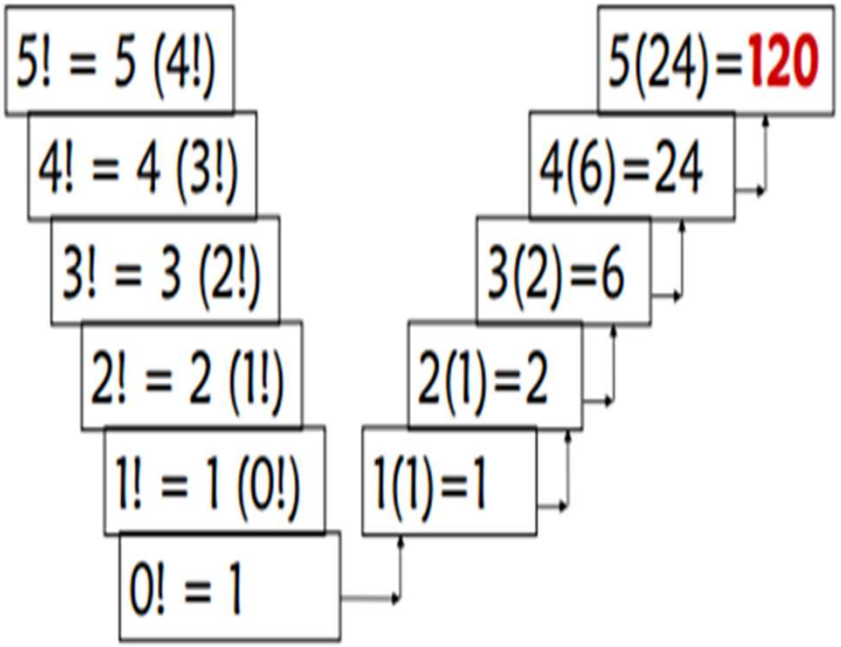
- De esta expresión es fácil ver que el factorial de n es también :

$$n! = n * (n - 1)!$$

Clase 2: TEMARIO. Recursividad

unab

Ejecución:




```
#area de funciones
def factorial(n):
    if(n==0):
        #print(1)
        return 1
    elif(n==1):
        #print(1)
        return 1
    else:
        x = n*factorial(n-1)
        #print(x)
        return x
```

Clase 2: TEMARIO. Recursividad

unab

```
1   # area de funciones
2   def factorial_normal(n):
3       r = 1
4       i = 2
5       while i <= n:
6           r *= i  # r=r*i
7           i += 1  # i=i+1
8       return r
9
10  #programa principal
11  v=int(input("ingresa un número mayor que
12  print (factorial_normal(v) )
13
```

```
#area de funciones
def factorial(n):
    if(n==0)or (n==1):
        #print(1)
        return 1
    else:
        x = n*factorial(n-1)
        #print(x)
        return x

#programa principal
resultado =factorial(5)
print(resultado)
```

Clase 2: TEMARIO. Recursividad

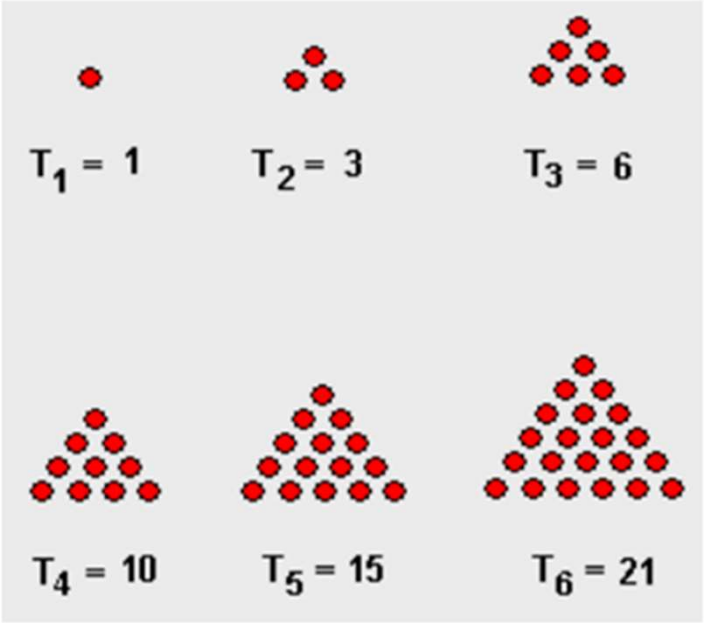
Otro Ejemplo:

Dícese que los Pitagóricos sentían una conexión mística con la serie de números **1, 3, 6, 10, 15, 21, ...**

Puedes encontrar el próximo miembro de esta serie?

El enésimo termino se consigue añadiendo **n** al termino previo

Los números en esta serie se llaman **triangulares** porque pueden ser visualizados de forma triangular



Clase 2: TEMARIO. Recursividad

unab

Pseudo-código:

- Si introducimos 4,
- 4 != 1 por lo tanto ejecuta 4 + triangulo(3)
- Entra la función con argumento 3
- 3 != 1 por lo tanto 3 + triangulo(2)
- Entra la función con argumento 2
- 2 !=1 por lo tanto 2 + triangulo(1)
- Entra la función con argumento 1
- 1 = 1 por lo tanto retorna 1
- Vuelve a la función con argumento 2
- triangulo(1) = 1, retorna 2 + 1 = 3
- Vuelve a la función con argumento 3
- triangulo(2) = 3, retorna 3 + 3 = 6
- Vuelve a la función original, con argumento 4
- triangulo(3) = 6, retorna 4 + 6 = 10

```
#area de funciones
def triangulares(t):
    if(t == 1):
        #print(1)
        return 1
    else:
        x = t+triangulares(t-1)
        #print(x)
        return(x)
#programa principal
resultado =(triangulares(6))
print(resultado)
```

Clase 2: TEMARIO. Recursividad

Python nos permite una declaración sencilla de **funciones recursivas**, solamente debemos "llamar a la función, dentro del cuerpo de la (misma) función".

Una definición correcta de una función recursiva, debera tener el cuenta lo siguiente:

•Existencia de un caso base (o inicial), la función retorna un valor

Manejo del caso base: Utilizando un if (se terminan los llamados recursivos).

Clase 2: TEMARIO. Recursividad

unab

Serie Fibonacci: En matemática, la sucesión de Fibonacci se trata de una serie infinita de números naturales que empieza con un 0 y un 1 y continúa añadiendo números que son la suma de los dos anteriores:

0, 1, 1, 2, 3, 5, 8, **13**, 21, 34, 55, **89**, 144, 233, 377, 610, 987, 1597…

Fibonacci (7) Debería retornar 13

Fibonacci (11) Debería retornar 89

Resolución: Se calcula el elemento **n** sumando los dos anteriores, es decir (**n**-1) + (**n**-2). Se asume que los dos primeros elementos son 0 y 1.

Clase 2: TEMARIO. Recursividad

unab

```python
#area de funciones
def fibonacci_iterativo(n):
    a = 0
    b = 1
    fib = 0
    if(n == 0)or (n == 1):
        fib = n
    else:
        #n es mayor igual a 2
        i = 2
        while(i <= n):
            fib = a+b #sumamos los valores
            a = b #actualizamos el valor de a
            b = fib #la suma de los anteriores
            i = i+1
    return fib
#programa principal
resultado = fibonacci_iterativo(11)
print(resultado)
```

```python
#area de funciones
def fibonacci_recursivo(n):
    if(n==0):
        return 0
    elif(n==1):
        return 1
    else:
        return fibonacci_recursivo(n-1)+fibonacci_recursivo(n-2)
#programa principal
resultado = fibonacci_recursivo(11)
print(resultado)
```

Algoritmo y Estructura de datos

- Un **algoritmo** es un procedimiento paso-a-paso para realizar alguna tarea en un tiempo finito .
  - Por lo general, un algoritmo toma datos de entrada y produce una salida basada en él.

- Una **estructura de datos** es una forma sistemática de organizar y acceder datos.

Clase 2: TEMARIO.
Introducción TDA

Tipos Abstractos de Datos:

- Un **Tipo Abstracto de Datos** (TDA) es la representación de una estructura de datos y las operaciones permitidas para dicha estructura.

- El uso de una interfaz a través de la cual es posible realizar las operaciones permitidas, **abstrayéndonos** de la manera en cómo estén implementadas dichas operaciones.

- El **encapsulamiento** es la propiedad de los objetos de permitir el acceso a su estado únicamente a través de su **interfaz** o de relaciones preestablecidas con otros objetos.

Clase 2: TEMARIO.
Introducción TDA


```python
class Empleado:
    def __init__(self, nombre, edad, dni):
        self.nombre = nombre
        self.edad = edad
        self.dni = dni

    def __str__(self):
        cadenaPrint=self.nombre+","+str(self.edad)+","+str(self.dni)
        return cadenaPrint



#programa ppal

empleado1=Empleado("Juan", 30, 12345678)
print(empleado1)
```

Clase 2: TEMARIO.
Introducción TDA


```python
# Ejemplo de una clase Humano

#Creamos la clase Humano
class Humano():
    #Definimos al Humano
    def __init__(self, edad, nombre, ocupacion):
        self.edad = edad
        self.nombre = nombre
        self.ocupacion = ocupacion
```

```python
#Creación de un nuevo método (función que imprime un mensaje)
def presentar(humano):
    presentacion = "Hola soy " + humano.nombre + " mi edad es " + str(humano.edad) + " y mi ocupación es " + humano.ocupacion #armo el mensaje
    print (presentacion)
```

```python
    #Creamos un nuevo método para cambiar la ocupación:
    #En caso que esta persona sea contratada

def contratar(humano, puesto): #añadimos un nuevo parámetro en el método
    #Ahora cambiamos el atributo ocupación
    humano.ocupacion = puesto
    print (humano.nombre + " ha sido contratado como " + humano.ocupacion)


# Programa Principal

personal = Humano(31, "Pedro", "Desocupado") #Instancia
presentar(personal) #Llamamos al método
contratar(personal,"Obrero")
presentar(personal)#Lo volvemos a presentar luego de su contratación
```

Clase 2

unab

Consultas

Clase 2: CIERRE DE LA CLASE

unab

Temas a desarrollar la próxima clase

- ❑ Listas
- ❑ Pilas
- ❑ Colas
- ❑ Árboles