MÓDULO

4

Área: **NEGOCIOS**

Curso: **PROGRAMACIÓN BÁSICA (PYTHON)**

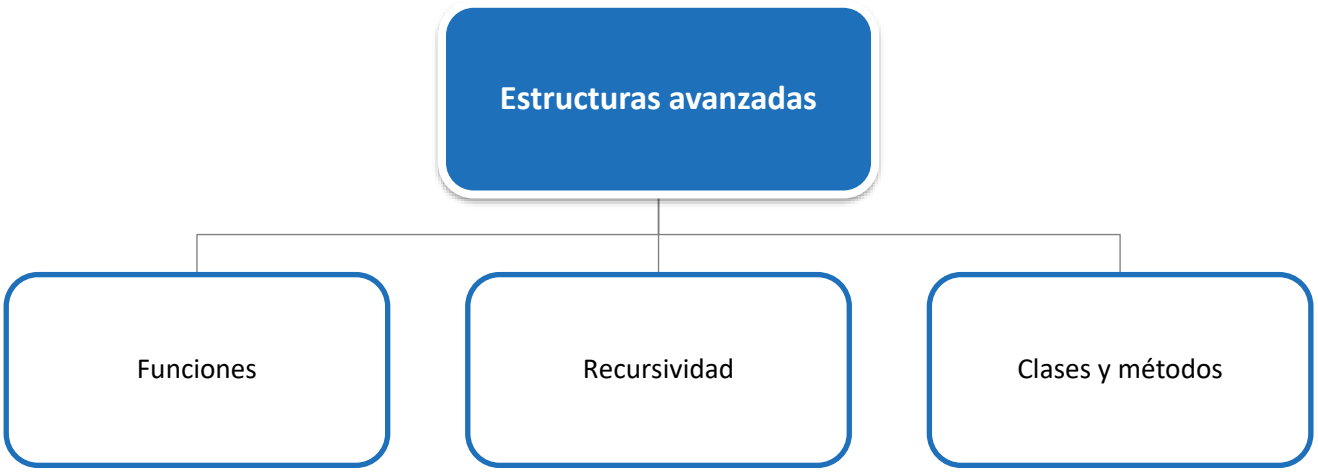
Módulo: Estructuras avanzadas

IPP

SUEÑA • APRENDE • CRECE

Índice

Mapa de Contenido



RESULTADO DE APRENDIZAJE DEL MÓDULO	Crea programas computacionales de mayor complejidad, utilizando estructuras avanzadas de datos y lenguaje, para representar la solución a un requerimiento escrito.

Introducción

En el mundo laboral enfocado en la programación se trabaja por módulos donde un programa es realizado por muchos programadores. En base a esto cada programador se encarga de un módulo de este programa, generando clases con métodos y funciones propias, con el fin de poder, posteriormente, importar todas las clases en un solo archivo y así poder ejecutarlo y obtener la información necesaria.

También este es muy utilizado cuando queremos reutilizar código, es decir, si somos unos programadores matemáticos y siempre tenemos que generar nuestras funciones de bisección, falsa posición, etc., nuestra mejor manera de trabajo es ir creando nuestra propia clase y después ir ejecutando códigos de ella, así nos despreocupamos, en cierto sentido, de que tenemos que pensar cómo funciona un método de bisección o cómo programarlo, solo nos enfocamos en la progresión del nuevo problema.

Desde la creación de la programación orientada a objeto (POO), la informática tuvo un cambio muy grande ya que antiguamente todo era programado en un solo archivo que era con millones de líneas de códigos y poder encontrar un error o realizar un cambio en ellos, contenía mucha complejidad, ahora las clases o los archivos de programación siempre se encuentran en rango que son muy cortos de leer, dado que separan en módulos los que son el core del programa o las funciones de este.

1. Funciones

1.1. Estructura

Una función es una estructura de python para agrupación de actividades con un objetivo determinado, se define mediante el comando “def” y se termina con “:” la declaración. Este puede ser ejecutado o utilizado de la misma manera que una variable, la diferencia que este puede o no puede retornar información. Una de lo más importante de una función son los argumentos, los cuales son variables u objetos que se le entregan a una función, para que esta trabaje en él.

Ejemplo:

```
def suma(arg_1, arg_2):
        c = arg_1 + arg_2
        print(c)
```

Esta función lo que realiza es mostrar por consola, la suma de 2 variables los cuales son entregados mediante 2 argumentos (arg_1 y arg_2), pero si nosotros queremos crear una función la cual nos retorne la suma y no la muestre por consola directamente, se ocupa el comando “return” ejemplo.

```
def suma(arg_1, arg_2):
        c = arg_1 + arg_2
        return c
```

De esta manera están definida las funciones de 2 maneras, una función que retorna información y otra que no, las funciones que no retornan información son conocidas como funciones void y las funciones que retornar información son del tipo de información que retornan, es decir, si yo retorno un número, mi función será definida como int o doublé, o si retorno un texto esta será definida como una función string, esto es importante dado que si retorno información mediante una función la variable que recibe esta información debe ser del mismo tipo que la función.

a. Cómo llamar una función

Las funciones que retornan información tienen que ser llamada por una variable, ejemplo:

```
Variable_a = suma(5, 6)
```

Y las funciones que no retornan nada pueden ser llamada directamente, como el print(),

Si se dan cuenta cuando ocupan el comando print() lo ponen directo en el código a diferencia que cuando ocupamos un comando para castear como str(texto), este tiene que ser igualado a una función.

Bueno ahora que tenemos funciones declaradas, el funcionamiento de mi programa es algo distinto, dado que tengo que avisarle cual es el núcleo del programa ahora a Python, esto se hace con una función main(), entonces la estructura será de la siguiente forma.

```
def funcion1():
        acciones()

def funcion2():
        acciones()

if __name__ == "__main__":
   #desde acá phyton comienza a ejecutar el programa de la manera que ahora nosotros conocemos.
```

DEFINICIÓN

Muy importante siempre tiene que estar todo identado.

Ejemplo:

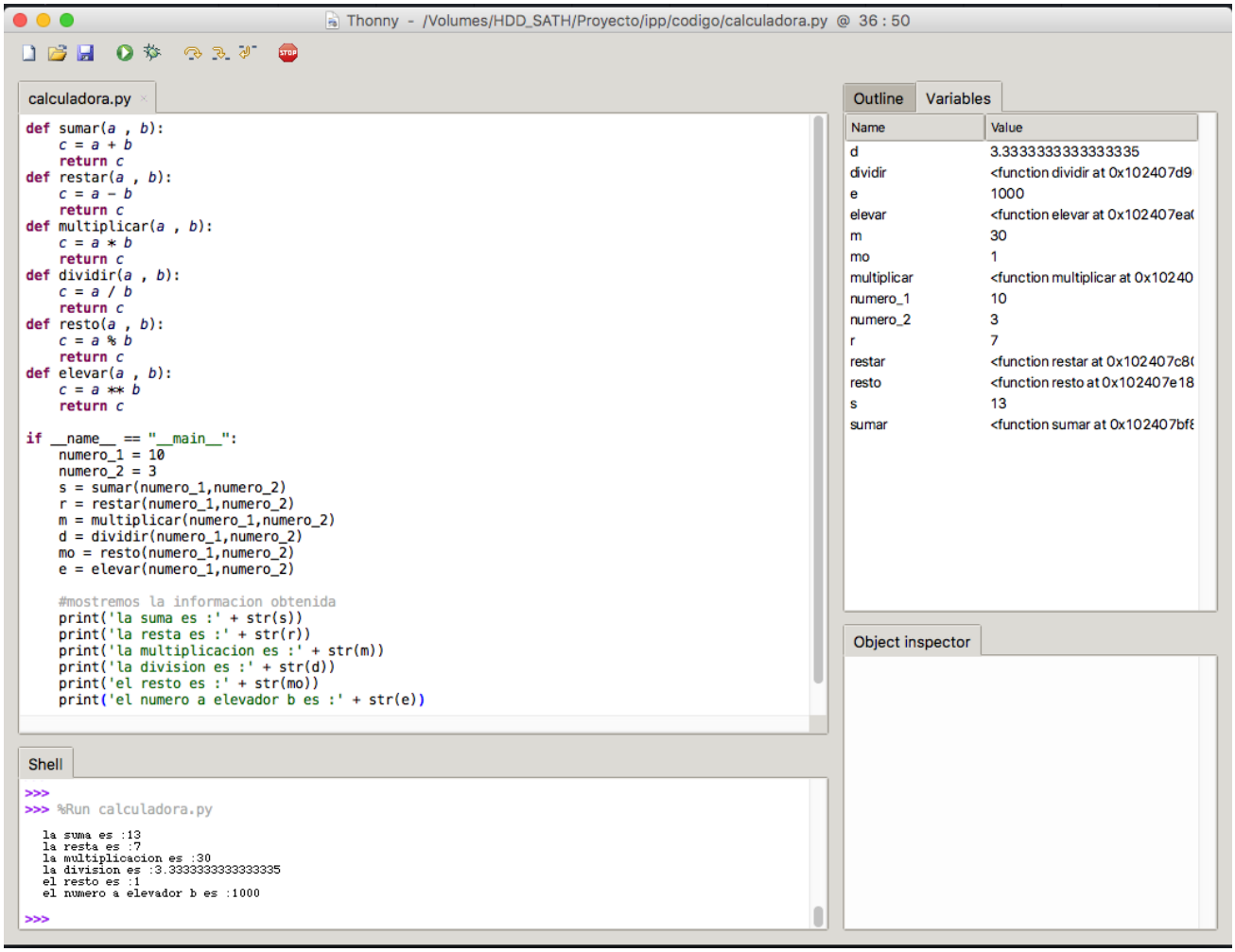
Crear una calculadora.

```
def sumar(a , b):
    c = a + b
    return c
def restar(a , b):
    c = a - b
    return c
def multiplicar(a , b):
    c = a * b
    return c
def dividir(a , b):
    c = a / b
    return c
def resto(a , b):
    c = a % b
    return c
def elevar(a , b):
    c = a ** b
    return c

#continua en el bloque siguiente ->
```

```
if __name__ == "__main__":
    numero_1 = 10
    numero_2 = 3
    s = suma(numero_1,numero_2)
    r = restar(numero_1,numero_2)
    m = multiplicar(numero_1,numero_2)
    d = dividir(numero_1,numero_2)
    mo = resto(numero_1,numero_2)
    e = elevar(numero_1,numero_2)

    #mostremos la informacion obtenida
    print('la suma es :' + str(s))
    print('la resta es :' + str(r))
    print('la multiplicacion es :' + str(m))
    print('la division es :' + str(d))
    print('el resto es :' + str(mo))
    print('el numero a elevador b es :' + str(e))
```

Si se dan cuenta en el ejemplo primero definimos las funciones, que reciben los argumentos necesarios y después ejecutamos todo en if name.

Vista en thonny el programa.



1.2. Estructura de una función

Dentro de una función se puede crear la cantidad de variables que se estime conveniente, en el caso anterior creamos en todas las funciones la variable c, pero esta variable es temporal, solo existe muestras de este ejecutando esa función, es decir que al momento de retornar una variable o termine su ejecución, la variable dejara de existir, las únicas variables que reconoce Python son los argumentos que se le entrega. Estos argumentos son solo copia de los datos:

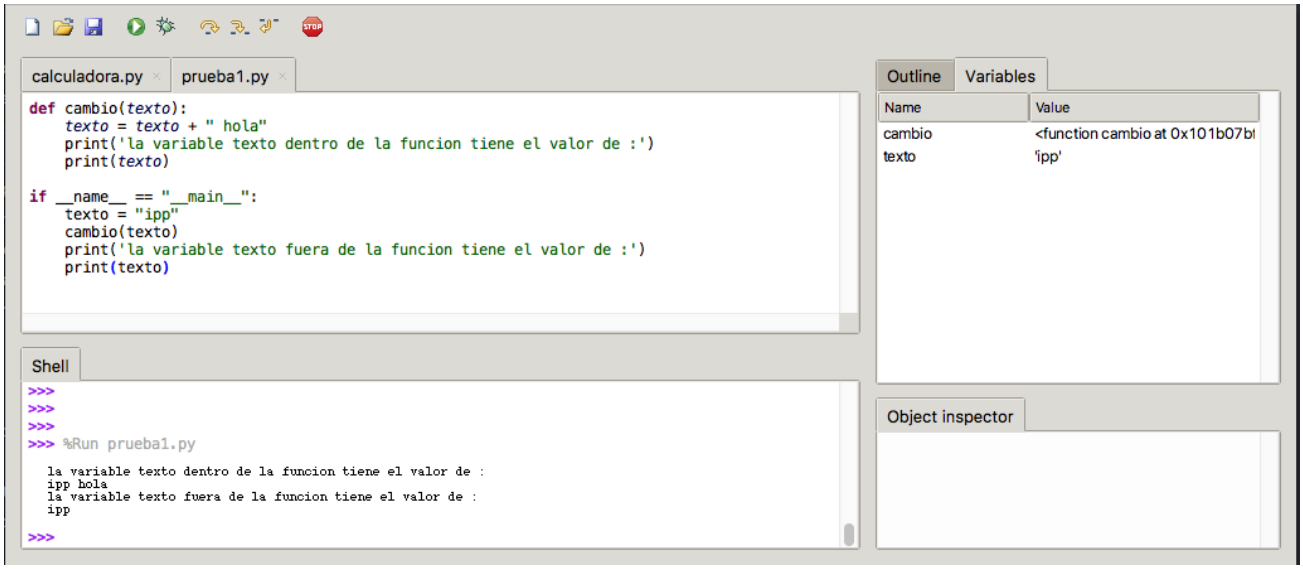
Ejemplo:

```
def cambio(texto):
    texto = texto + " hola"
    print('la variable texto dentro de la función tiene el valor de :')
    print(texto)

if __name__ == "__main__":
    texto = "ipp"
    cambio(texto)
    print('la variable texto fuera de la funcion tiene el valor de :')
    print(texto)
```

Al momento de ejecutar este programa la consola nos muestra lo siguiente:

```
la variable texto dentro de la funcion tiene el valor de :
ipp hola
la variable texto fuera de la funcion tiene el valor de :
ipp
```

Vista en thonny el programa.



Se dan cuenta que el cambio que se realizó en la función fuera de esta no se ve reflejado, esto ocurre porque solo es una copia del valor de la variable la que se entrega a la función.

1.3. Variables Globales

Las variables globales son conocidas por ser variables que existen en todo el código, y están pueden ser llamadas tanto desde una función o desde el mismo main, a diferencia de un argumento entregado a una función, si esta función realiza un cambio en una variable global, esta cambia para todo el programa.

Al código Python también se le debe informar que ocuparemos una variable global dentro de una función con el comando “global”.

La estructura de una variable global es ir antes de las funciones. Quedando de la siguiente forma.

- Variables Globales
- Funciones
- If main
- La ejecución del código.

Realizaremos el mismo ejemplo anterior, pero dejando el texto afuera como variable global.

```
texto_global = "ipp"

def cambio():
    global texto_global
    texto_global = texto_global + " hola"
    print('la variable texto dentro de la funcion tiene el valor de :')
    print(texto_global)

if __name__ == "__main__":
    cambio()
    print('la variable texto fuera de la funcion tiene el valor de :')
    print(texto_global)
```

La consola nos muestra lo siguiente:

```
la variable texto dentro de la funcion tiene el valor de :
ipp hola
la variable texto fuera de la funcion tiene el valor de :
ipp hola
```

Si se dan cuenta, el valor de la variable tanto dentro, como fuera de la función es el mismo.

2. Recursividad

DEFINICIÓN

En programación se le define recursividad cuando una función se llama a sí misma. Es decir, tenemos una función X y dentro de esta función x llama nuevamente a sí misma.

Un ejemplo aplicado a la vida real de algo recursivo son los espejos en los ascensores, en los cuales cuando uno mira un lado de este y este tiene otro espejo en frente te das cuenta que la imagen se repite infinitas veces hasta el infinito.

Realicemos un ejemplo.

Recuerdan que cuando aprendimos ciclos programamos una función que era la función Fibonacci, bueno ahora realizaremos la misma función, pero de forma recursiva.

Para recordar, una serie Fibonacci es cuando para obtener el número siguiente de la serie se suman sus 2 números anteriores. Partiendo la serie con un 1 1.

Ahora les mostrare una función Fibonacci escrita sin recursión:

Pero esto nos da a pensar que si ocupamos una función recursiva el programa se nos ira al infinito y este nunca parará. Acá viene la importancia de la condición de término.

IMPORTANTE

Es importante poder generar una condición de que cuando se ingrese un valor a la función y esta se cumpla en vez de llamarse nuevamente, esta se retorne o termine, y en ese instante se comienza a devolver todo el código cerrando cada función que se abrió de sí mismo.

```
def fibonacci(num):
    a0 = 0
    a1 = 1
    fib = 0
    if num == 0:
        return 0
    if num == 1:
        return 0
    for i in range(2,num):
        fib = a0 + a1
        a0 = a1
        a1 = fib
    return fib
```

Ahora la serie Fibonacci escrita con recursión:

```
def fib(num):
    if num <= 1:
        return num
    else:
        return fib(num - 1) + fib(num - 2)
```

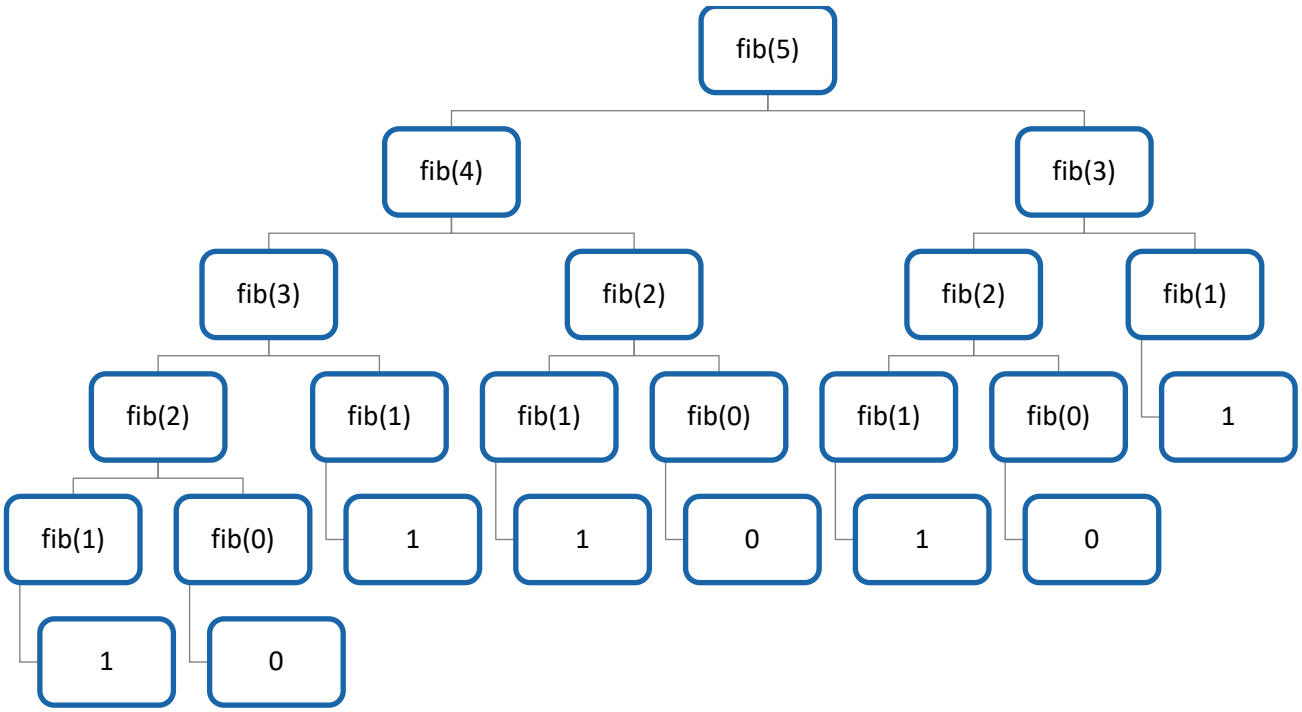
Como funciona este código, es bien sencillo, pongamos como ejemplo que entregamos el número 5.

Paso a paso

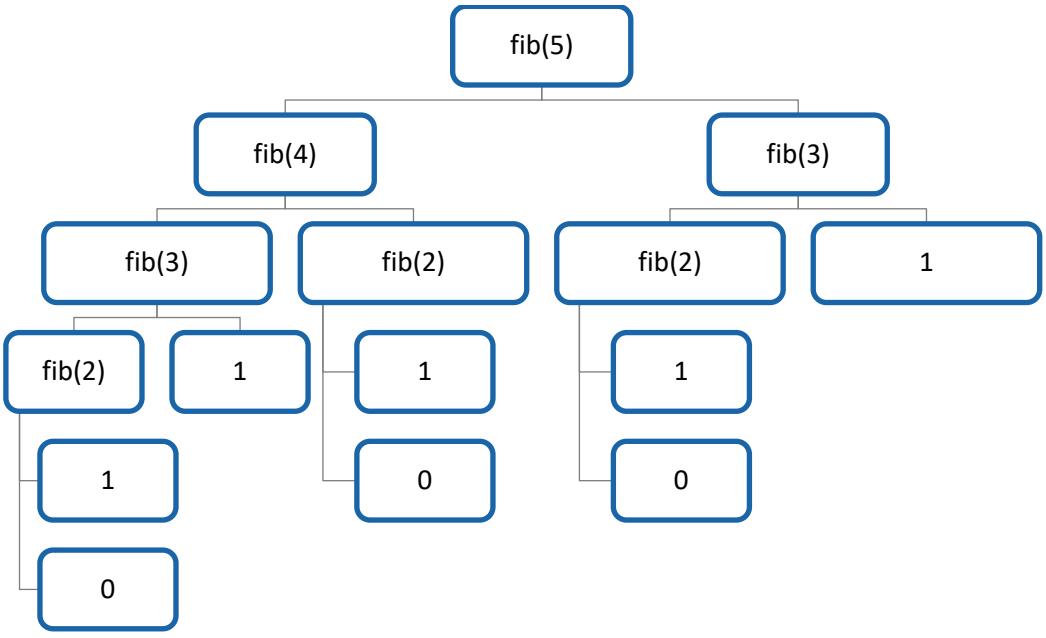
fib(5)
Primero verificamos si este número es menor o igual a 1.

En este caso no es así, por lo tanto pasamos al else, este dice que ahora retornaremos
fib(5 - 1) + fib(5 - 2)

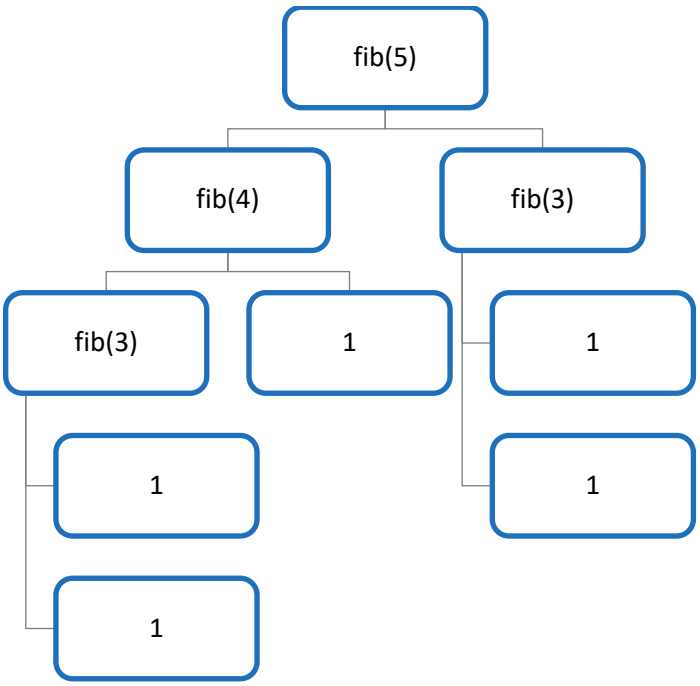
Ahora en conjuntamente se ejecutar el fib(4) y el fib(3) y dentro de cada uno se ejecuta fib(3) y fib(2) y del otro se ejecuta fib(2) y fib(1) y asi avanza. Hasta que en algún momento uno llega a 1 o menor, en ese caso retorna el número y ahí queda a la espera.



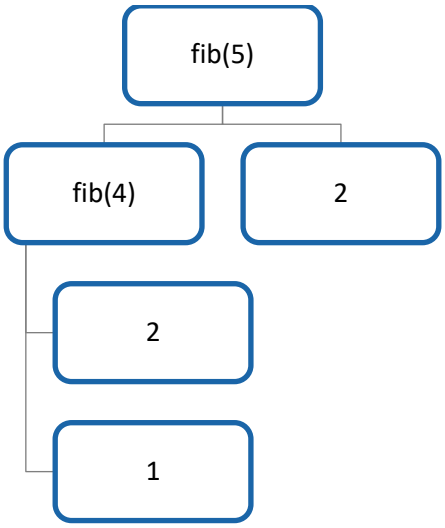
Así queda la secuencia, Después que llega al final se empieza a recoger o a devolverse.



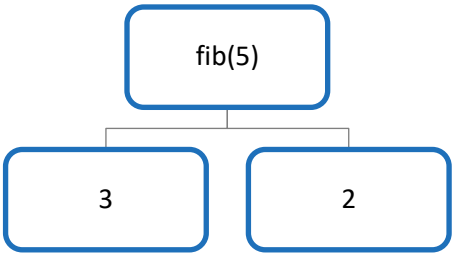
En la siguiente iteración queda:



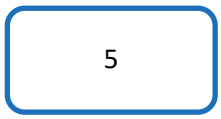
En la siguiente iteración queda:



En la siguiente iteración queda:



Y al final queda:



De esta manera se ve cómo funciona una recursión.

Otro ejemplo:

Crear una función mediante recursividad que pueda retornar la factorial de un número que se le entrega como parámetro (argumento).

Factorial de 5 = 1*2*3*4*5

Factorial de 0 es igual 1 no olviden eso y que también solo pueden ser numero naturales positivos.

Entonces este ejemplo recursivamente queda:

```
def fac(num):
   if num <= 1:
      return 1
   else:
      return num * fac(num-1)
```

Entonces si ejecutamos fac(5) la respuesta será de 120.

3. Clases y métodos

3.1. Clases

Una clase en Python es la característica de programación de un objeto, una estructura que tendrá sus propias funciones y también datos. Esta se va creando y generando en tiempo de ejecución y puede modificarse luego de su creación.

Las clases son definidas por el comando class, y tiene la siguiente estructura:

```
class monbre_clase:
        declaración_1:
        declaración_2:
```

Las declaraciones de una clase pueden ser variables globales, métodos o funciones.

Creemos un ejemplo de una clase:

```
class mi_clase:
    a = "este es un texto"
    b = 12

    def fun(self):
        return "hola ipp"

if __name__ == "__main__":

    variable_a = mi_clase()
    print(variable_a)
    print(variable_a.a)
    print(variable_a.b)
    p = variable_a.fun()
    print(p)
```

La consola nos mostrara lo siguiente:

```
<__main__.mi_clase object at 0x1022d6e80>
este es un texto
12
hola ipp
```

En las clases existe un método que se llama init, este método es el que inicia la clase. Se ocuipa de la siguiente manera.

def __init__(self, arg_1, arg_2, …):

Es muy importante el objeto self, este tiene que ir en todas las funciones de nuestra clase.

Iniciemos un método de inicio de nuestra clase:

Ahora crearemos una clase que me tenga un objeto de un punto en el espacio en un plano de 3 Dimensiones.

```
class punto:
    def __init__(self, arg_1, arg_2, arg_3):
        self.x = arg_1
        self.y = arg_2
        self.z = arg_3
    def vect(self):
        return [self.x,self.y,self.z]

if __name__ == "__main__":

    variable_a = punto(10,20, 3)
    print(variable_a.x)
    print(variable_a.y)
    print(variable_a.z)
    print(variable_a.vect())
```

Al ejecutarlo nos muestra lo siguiente:

```
10
20
3
[10, 20, 3]
```

Ahora creemos otro ejemplo algo más conocido, una mascota de tipo perro.
Entonces tenemos que pensar qué es lo que tiene todos los perros en común y qué es lo que puede variar.

Lo que tiene en común que todos son de caninos, y la cantidad de patas que tiene (pensado que todos los perros siempre nacerán y tendrán 4 patas). Ahora vemos lo que particular de cada perro, nombre, pelaje, tamaño, peso. Etc.

Entonces creemos la clase perro.

```
class perro:
    tipo = 'canino'
    patas = 4
    def __init__(self, nombre, pelaje, tamano, peso):
        self.nombre = nombre
        self.pelaje = pelaje
        self.tamano = tamano
        self.peso = float(peso)
    def alimentar(self):
        self.peso += 1

if __name__ == "__main__":

    a = perro('firulais', 'largo', 'medio', 4.6)
    b = perro('max', 'corto', 'grande', 20)

    print(a.nombre)
    print(b.nombre)

    b.alimentar()
    print(b.peso)
```

Al ejecutarlo por consola nos muestra lo siguiente:

```
firulais
max
21.0
```

Es muy sencillo crear una clase, pero fíjense que en este ejemplo ocurrió algo curioso que los nombres del argumento entregado por el init eran iguales que los nombres de los objetos de las variables de la clase, entonces Python como identifico cual es cual.

Python pudo identificar cual es la variable de la clase y cual era del argumento por la función self, self es un objeto perteneciente a la clase por este motivo siempre que queramos modificar una información dentro de la clase esta debe ser self y punto la variable de esta, como también esta como regla que todas las funciones de la clase tengan el objeto self.

IMPORTANTE

Se puede crear un archivo .py en el cual solo este una clase y sin main, y después en un archivo .py completamente distinto ocupar la clase ya creada, para esto se necesita importar.
Import nombre_clase
el nombre de la clase va sin el .py y este debe estar en la misma carpeta que el archivo del cual se llama.

3.2. Métodos especiales

Al igual es __init__, es un método creado para iniciar la clase existen un tipo de métodos especiales en Python los cuales son muy utilizados:

__str__ : Es utilizado cuando se quiere realizar una conversión del objeto a string, dentro de esta función se le entregara la forma que debe retonar el objeto.
__add__: es la forma que se comportara al sumar 2 objetos o 2 clases.
__sub__: es la manera que se comportara al restar 2 objetos o clases.
__call__: es cuando se llama a la variable de la clase con paréntesis.

Como estas mencionadas existe una gran cantidad de sentencias conocidas como métodos especiales, pero las más utilizada es init y call, el resto dependerá del tipo de problema que deseen aplicar.

3.3. Librerías

En Python existen una diversidad de librerías tanto de Python como de terceras personas, Las librerías se ocupan para acciones específicas, como el uso de un entorno grafico o también poder usar formulas trigonométricas. Estas nos pueden ser muy útiles en todo tipo de situaciones.

Las librerías son importantes impórtalas, estas como aprendimos en las normas PEP-8 Siempre van al inicio de nuestro código Python.

Si queremos saber de alguna librería en específico toda se encuentra en la documentación oficial de la página de Python. Que es https://docs.python.org/3/library/index.html

Ahora veremos cómo importar una de las librerías más utilizadas en Python para unos estudiantes de Ingeniería que es la librería math, que es una librería matemática que contiene funciones trigonométricas, cálculos de variables, integrales entre otras cosas.
Algunos de los comandos se encuentran en la documentación https://docs.python.org/3/library/math.html

Ej:
Crear una función que muestre de coseno desde 1 hasta 10.

```
import math
def fun1(x):
   return math.cos(x)
if __name__ == "__main__":
   for i in range(11):
      aux = fun1(i)
      print(aux)
```

Ocupando la materia aprendida lo que realizamos es importar primero la librería, después definimos una función con el nombre fun1 que esta retornar el resultado, en este retorno viene lo más importante, yo al momento de ocupar un módulo de una librería siempre que no se nos olvide debe ir antes el nombre de la librería acompañada de un punto y la función específica de esta, esto trabajo de igual manera cuando creamos una clase solo que no es necesario declarar una variables para poder usar las funciones.

Dentro del main realizamos que llame a la función fun1 con números de entrada desde 0 hasta 10, esto nos muestra lo siguiente:

```
1.0
0.5403023058681398
-0.4161468365471424
-0.9899924966004454
-0.6536436208636119
0.2836621854632263
0.9601702866503661
0.7539022543433046
-0.14550003380861354
-0.9111302618846769
-0.8390715290764524
```

Los resultados son los esperados, según la documentación los cálculos trigonométricos se encuentran en radianes.

Todas las librerías trabajan exactamente de la misma manera. Si existe una librería que ustedes encontraron documentación y no es estándar de Python, esta debe ser instalada, para eso existe una herramienta llamada pip, la cual instala librerías de Python en nuestro computador, esta no es materia del curso, pero si les ayudaría mucho saber que existe en caso que quieran usar una librería que no existiera.

7. Cierre

Al momento que se creó la programación orientada a objetos (POO) se abrió un mundo de posibilidades en la programación generando una reutilización de código de manera considerable, antes sin POO los códigos eran muy largos y con mucha dificultad de poder encontrar error, saber si se está trabajando con una función de manera innecesaria o está mal ejecutada, cuando POO apareció todo esto se mejoró dado que los programadores generaban sus propias clases para cada tipo de necesidad y posteriormente solo la volvían a ocupar en algún otro proyecto y así mismo estas funciones constantemente se iban mejorando o añadiendo nuevas funciones.

Ahora en la actualidad en el año 2017, se ocupan esto para crear framework que son estructuras de trabajo, pudiendo generar sistemas web, aplicaciones de sistemas de comunicaciones, seguridad entre otros. En este módulo hemos aprendido cómo crear una primera clase dando un primer paso a este gran mundo de la Programacion Orientada a Objetos (POO).