

GUÍA DE APRENDIZAJE

OPERADORES de asignaciones

```
# ejercicio de operador -=  
# resta una cantidad de  
# la variable nnn  
nnn = 52  
nnn -= 18  
print(nnn)
```

aritméticos

```
# ejercicio de operador +  
# suma variables  
nnn = 52  
ppp -= 18  
print(nnn+ppp)
```

Cree variables y aplique el operador -=, como lo muestra la imagen.

Haga lo mismo con los operadores +=; ==; *=; /=; **=; //=; %=.

Analice que la línea que comienza con numeral es código de referencia, es para hacer anotaciones que nos explican que hicimos.

Cree variables y aplique el operador +, como lo muestra la imagen.

Haga lo mismo con los operadores: -; *; /; **; //; %.

Analice que la línea que comienza con numeral es código de referencia, es para hacer anotaciones que nos explican que hicimos.

relacionales

```
# ejercicio de operador ==  
# define igualdad entre variables  
nnn = 52  
ppp = 18  
print(nnn == ppp)
```

Cree variables y aplique el operador ==, como lo muestra la imagen.

Haga lo mismo con los operadores: !=; >; <; >=; <=.

Analice que la línea que comienza con numeral es código de referencia, es para hacer anotaciones que nos explican que hicimos.

Ahora realiza este ejercicio, en el cual se busca averiguar el tipo de variable contenida, el nombre de la variable no puede ser la misma siempre, debes cambiarla en cada línea. Para la variable booleana un operador de relación.

```
# ejercicio descubrir tipo de dato  
# de una variable  
  
print("variable numero entero")  
nnn = ###  
print(nnn, type(nnn))
```

```
print("variable numero decimal")  
nnn = ###  
print(nnn, type(nnn))
```

```
print("variable numero imaginario")  
nnn = ###j  
print(nnn, type(nnn))
```

```
print("variable cadena de texto")  
nnn = ###  
print(nnn, type(nnn))
```

```
print("variable booleana true - false")  
nnn = ###  
print(nnn, type(nnn))
```

```
palabra = input("Introduce una palabra: ")  
num_int = int(input("Introduce un número entero: "))  
num_float = float(input("Introduce un número flotante: "))  
num_complex = complex(input("Introduce un número complejo: "))
```

```
print("String: ", palabra)  
print("Entero: ", num_int)  
print("Flotante: ", num_float)  
print("Número complejo: ", num_complex)
```

INPUT:

```
2 # DATO TIPO FLOAT  
3 real = 1.1 + 2.2  
4 # MUESTRA TODO EL DECIMAL  
5 print(real)  
6 # MUESTRA SOLO DOS DECIMALES  
7 print(f'{real:.2f}')  
8  
9 # DATO TIPO COMPLEX  
10 # <parte_real>+<parte_imaginaria>j  
11 DD = 25 + 3j  
12 # MUESTRA LA PARTE REAL  
13 DD.real  
14 # MUESTRA LA PARTE IMAGINARIA  
15 DD.imag
```

GUÍA DE APRENDIZAJE

Realiza este mismo ejercicio, pero para realizar una potencia, y si dos números son diferentes.

```
nombre = input("¿Cual es tu nombre?: ")
print("Hola " + nombre + ", vamos a realizar una suma.")

num_uno = int(input("Porfavor introduce el primer valor: "))
num_dos = int(input("Porfavor introduce el segundo valor: "))

resultado = num_uno + num_dos

print(nombre + " el resultado de la suma es: ", resultado)
```

EJERCICIO IF (SI CONDICIONAL)



QUIERO HACER UN PROGRAMA PARA DIGITAR LA NOTA DE MATEMÁTICAS. PERO QUE **Si** ESCRIBO UN VALOR MAYOR A 5 ME SALGA UN MENSAJE DE ERROR

PRIMERO: QUE DESEAMOS HACER:

- ☐ ESCRIBIR LA NOTA DESDE EL PROGRAMA.
- ☐ QUE EL USUARIO DIGITE LA NOTA.

SEGUNDO: SEGÚN LO QUE ESCOGISTE EN EL PRIMER PASO AHORA DEBEMOS:

- ☐ CREAR UNA VARIABLE SIMPLE: NOTA = 5
- ☐ CREAR UNA VARIABLE CON UN INPUT



¡¡¡ ATENCION!!!

RECUERDA, PARA QUÉ ES EL NÚMERO, ES UNA VARIABLE QUE ALMACENE NÚMEROS DECIMALES.

EL PROGRAMA SE VE ASÍ:

```
variable
```

```
if condición:
    print("el
```

```
if condición:
    print("el
```

EL IF (SI) DEBE TERMINAR CON DOS PUNTOS.


LA SIGUIENTE DEBE IR A UN TABULAR DEL BORDE.


GUÍA DE APRENDIZAJE

EN PYTHON LAS CONDICIONES SIGUEN ESTA NORMA:



Pongamos este ejemplo, MARTA tiene una hija de nombre EUDORA, y JORGE tiene un hijo de nombre MARCOS, ellos se conocieron y formaron una familia.

NOTE que MARTA y JORGE están al inicio del borde ya que ellos son comandos TERMINAN EN .

NOTE que EUDORA se escribe con un  de distancia del borde, porque ella es producto del comando MARTA, lo mismo le sucede a MARCOS.

PERO AHORA TODOS FORMARON UNA FAMILIA. ASÍ QUE TODOS PERTENECEN A UN MISMO PROGRAMA.



YA SABES QUE HACER. SIGUE CON MUCHO CUIDADO LAS INDICACIONES Y CREA UN PROGRAMA EN PYTHON.

PRIMERO: Que se IMPRIMA en pantalla:

Sistema para calcular el promedio de un Estudiante.

SEGUNDO: cree una variable con un INPUT que pregunte: **Para comenzar, ¿Cuál es tu nombre?:**

TERCERO: cree una variable de nombre una asignatura cualquiera, en ella cree un INPUT QUE SOPORTE decimales y que diga: **¿Cuál es tu calificación en matemáticas?**

EJEMPLO: Lo de abajo se lee:

Si la variable **AMIGO** es igual a 5 se mostrará un letrero que dice **"copiaste el número correcto"**, **SI NO ENTONCES "copiaste el número incorrecto"**

```
amigo = 5

if amigo = 5:
    print("copiaste el numero correcto")
else:
    print("copiaste el numero incorrecto")
```

if SE LEE: SI...
else SE LEE: SI NO, ENTONCES...

CUARTO: Realice tres veces más el paso **TRES**.

QUINTO: Cree una variable de nombre **DEFINITIVA**, en ella calcule el promedio de las asignaturas.

SEXTO: deje claro que la variable **DEFINITIVA** acepta decimales.

SÉPTIMO: escriba en lenguaje Python la siguiente oración:

Si la variable definitiva es mayor o igual a 3, entonces **print('Felicidades ' + '???' + ' "aprobaste" con un promedio de: ', DEFINITIVA)**

Si no es así usa la palabra **else:**

print("Lamento informarle que debes recuperar " + '???' + ' "no aprobaste" tu promedio es de: ', DEFINITIVA)

SÉPTIMO: corra el modulo y rectifique.

GUÍA DE APRENDIZAJE *Vamos a practicar...*

```
4 if nn == pp or pp >= 6:
5     print("")
6 else:
7     print("")
```

```
if nn == pp and pp >= 6:
    print("")
else:
    print("")
```

```
4 if not nn == pp:
5     print("")
6 else:
7     print("")
```

ELIF
AGREGA MÁS
CONDICIONES

EJERCICIO 1:

```
ESCRIBE TU EDAD: 15
NO ERES MAYOR DE EDAD

=====
ESCRIBE TU EDAD: 21
YA ERES MAYOR DE EDAD
```

EJERCICIO 2:

realiza e
investiga que
hace el método
lower()

EJERCICIO 3:

Investiga que hace el operador %. Ahora cree dos variables tipo número entero, para introducir datos desde el teclado. Ahora lee la premisa y conviértala en lenguaje Python 3:

Si entre las variables el residuo es igual a cero, que se imprima un texto así: LA DIVISIÓN ES EXACTA EL COCIENTE ES ###. Si no, entonces que se imprima LA DIVISIÓN ES INEXACTA, EL COCIENTE: ## Y EL RESIDUO: ###.

```
key = "contraseña"
password = input("Introduce la contraseña: ")
if key == password.lower():
    print("La contraseña coincide")
else:
    print("La contraseña no coincide")
```

EJERCICIO 4:

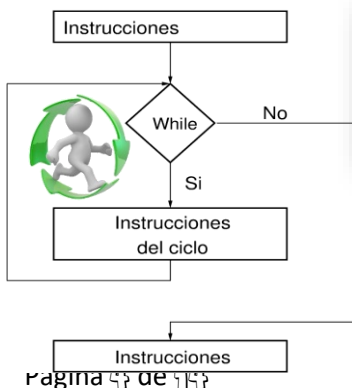
Investiga que hace el operador %. Ahora cree UN PROGRAMA QUE DIGA SI UN NÚMERO ES PAR O IMPAR USANDO **if** Y **else**.

EJERCICIO 5: La DIAN establece cada año unos topes económicos que indican si una persona natural debe **DECLARAR RENTA** o no. Vamos a hacer un ejercicio para que puedas descubrir si entras en el selecto grupo de personas que tienen que declarar su renta este año.

- El usuario debe contestar esta pregunta con una S para SI o con una N para NO
- ¿Tuviste ingresos superiores a \$49.850.000 durante todo el 2021?
- Si contesta no que aparezca un mensaje: NO DEBES PAGAR IMPUESTO A LA RENTA.
- Si contesta que sí, que le pregunte cuanto ganó y le calcule el valor a pagar sobre el 1,2% anual.

EJERCICIO 6: Analice para qué sirve el siguiente ejemplo y defina qué pasa en cada línea de comando.

```
name = input("¿Cómo te llamas? ")
gender = input("¿Cuál es tu sexo (M o H)? ")
if (gender == "M" and name.lower() < 'm') or (gender == "H" and name.lower() > 'n'):
    group = "A"
else:
    group = "B"
print(f"Tu grupo es {group}")
```



```
5 while CONDICIÓN A CUMPLIR:
6     print()
7     PROCESO A REALIZAR
8 else:
9     LO QUE PASA CUANDO YA
10    NO SE CUMPLA
```

EJERCICIO 1:

Si
deseáramos ver números
consecutivos enteros hasta el 10.

EJERCICIO 1A:

lo mismo que el ejercicio 1 pero deseamos que el usuario ponga el límite.

EJERCICIO 1B: Ahora que solo aparezca números pares, pero deseamos que el usuario ponga el límite.

F-STRINGS

GUÍA DE APRENDIZAJE

Listas []

`<lista>.append(<elemento>)`

La lista que será modificada

Método

Elemento que será agregado al final de la lista

DE IZQUIERDA A DERECHA DATO 0, DATO 1, DATO 2...
DE DERECHA A IZQUIERDA DATO -1, DATO -2...



Algunas propiedades de las listas:

Son **ordenadas**, mantienen el orden en el que han sido definidas

Pueden ser **formadas** por tipos arbitrarios

Pueden ser **indexadas** con `[i]`.

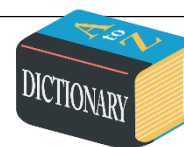
Se pueden **anidar**, es decir, meter una dentro de la otra.

Son **mutables**, ya que sus elementos pueden ser modificados.

Son **dinámicas**, ya que se pueden añadir o eliminar elementos.

```
2 'CREE UNA LISTA PARA TRABAJAR'
3 dd = [dato, dato, dato, dato]
4 print(dd)
5 'PRIMER DATO ES NUMERO 0'
6 print(dd[num dato])
7 dd[3] = otro dato
8 del dd[3]
9 dd += [dato, dato]
10 dd.append(dato)
11 dd.extend([dato, dato])
12 dd.insert(dato, posicion)
13 dd.remove(dato)
14 dd.pop()
15 dd.reverse()
16 dd.sort()
17 dd.index(dato)
```

CREE una **variable de lista** y aplique todos los métodos descritos, trate de usar datos de la tabla (**nativos**) y datos asignados por **usuario**.



Un **diccionario {}** es una colección de elementos, donde cada uno tiene una **llave (Key)** y un **valor (value)**.

Las **tuplas (tuples)**

son colecciones de datos idénticos o distintos clasificados con un índice y que **no** pueden ser **modificados**.

```
2 'CREE TUPLA PARA TRABAJAR'
3 dd = (dato, dato, dato, dato)
4 dd = dato, dato, dato, dato
5 print(dd)
6
7 ee = []
8 tupla = tuple(ee)
9 print(tupla)
10
11 print(dd.count(dato))
12 print(dd.index(posicion))
13 print(dd.index(posicional, posbuscada))
```

Son **dinámicos**, pueden crecer o decrecer, se pueden añadir o eliminar elementos.

Son **indexados**, los elementos del diccionario son accesibles a través del **key**.

Y son **anidados**, un diccionario puede contener a **otro diccionario** en su campo **value**.

Un **conjunto = set {}** es una colección **no ordenada** de objetos únicos. Python provee este tipo de datos «por defecto» al igual que otras colecciones más convencionales como las listas, tuplas y diccionarios. Los **elementos** de un set:

Son **únicos**.

Son **desordenados**.

Deben ser **inmutables**.

```
7 print(dd)
8 print(dd["clave#"])
9 dd["clave#"] = otro valor
10 dd.clear()
11 dd.copy()
12 dict.fromkeys(dd, valor)
13 dd.get("clave3")
```

```
2 #MODOS CREAR DICCIONARIOS
3 dd = {"clave1": "valor1",
4       "clave2": "valor2",
5       "clave3": "valor3"}
6 dd = dict([(tuple1),
7            (tuple2),
8            (tuple3),])
9 d3 = dict(clave1 = valor1,
10          clave2 = valor2,
11          clave3 = valor3)
```

```
2 x = {"ge", "a", "b", "c", "a"}
3 d = set("abca")
4 y = {"g", "m", "a"}
5 y.add("w")
6 print(y.difference(x))
7 print(x.difference_update(y))
```

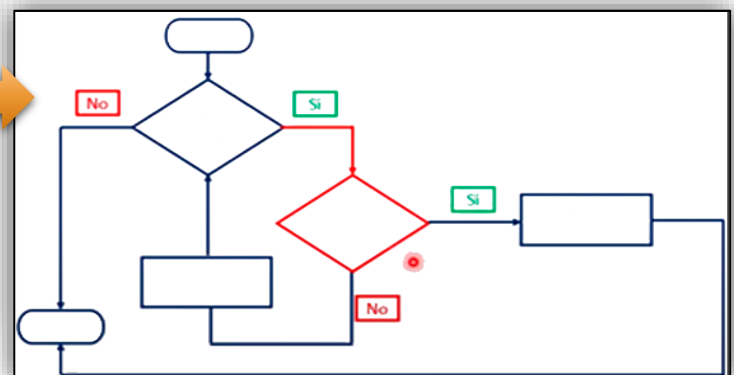
GUÍA DE APRENDIZAJE

add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two or more sets
intersection update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric difference()	Returns a set with the symmetric differences of two sets
symmetric difference update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with another set, or any other iterable

```
1
2 for <elem> in <iterable>:
3     <Tu código>
```

```
1 print("while con la sentencia break \n")
2 contador = 0
3 while contador < 10:
4     contador += 1
5
6     if contador == 5:
7         break
8
9     print("Valor actual de la variable: ", contador)
10
11 print("Fin del prgrama, la sentencia break se ha ejecutado.")
12
13 #Ejemplo para continue
14
15 print("\nwhile con la sentencia continue \n")
16 contador = 0
17 while contador < 10:
18     contador += 1
19
20     if contador == 5:
21         continue
22
23     print("Valor actual de la variable: ", contador)
```

El **BUCLE FOR** se utiliza para recorrer los elementos de un objeto **ITERABLE** (lista, tupla, conjunto, diccionario, ...) y ejecutar un bloque de código. En cada paso de la iteración se tiene en cuenta a un único elemento del objeto iterable, sobre el cuál se pueden aplicar una serie de operaciones.



`range(start, stop, step)`

OBLIGATORIO

CONCATENA CÓDIGOS PARA VER UNA TABLA DE MULTIPLICAR TENIENDO EN CUENTA QUE EL USUARIO DEBE DEFINIR CUÁL.

ANALIZA ESTE **Nested Loops**

Analiza este ejemplo de la serie de Fibonacci:

```
1 num_uno, num_dos = 0, 1
2 while num_dos <= 1597:
3     print(num_uno, num_dos, end=" ")
4     num_uno = num_uno + num_dos
5     num_dos = num_uno + num_dos
```

GUÍA DE APRENDIZAJE

```
19 # CALCULAR TABLA DE MULTIPLICAR
20 D = VARIABLE DE TABLA
21 for VARIABLE A MOSTRAR in range(DEFINA PARAMETROS):
22     print(f"{{?}} x {{?}} = {{????}}")
23
24 print("????")
```

PYTHON tiene **módulos** completos para ejecutar diversas tareas, ejemplo de ello es **random** y una de sus funciones **randint()** copia el código y entiende su proceso, luego trata de hacer un **emulador de dado**:

```
1 import random
2 go = random.randint(0, 1)
3 if go == 0:
4     print("SALE JUGADOR 1")
5 else:
6     print("SALE JUGADOR DOS")
```

```
25 TT = ["????", "????", "????", "????", ]
26 QQ = ["!!!!", "!!!!", "!!!!"]
27
28 for x in TT:
29     for y in QQ:
30         print(x, y)
```

Crea un lance de dados con
random.randint()

else: es la condición cuando el bucle se cumple.
Break detiene el bucle en un punto definido.

Vamos a realizar un pequeño juego, quien ataca se decide aleatoriamente, y cada jugador tiene un arma que solo puede usar 3 veces.



Soy un guerrero, lucharé con quien quiera perder, mi cuchillo es capaz de quitar 10 puntos de vida a mi oponente, y mi espada 15.

Soy tu contrincante, luchare en nombre de la PC, mi Lazo quita 12 puntos de vida a mi oponente, y mi espada 15



GUÍA DE APRENDIZAJE

```
NN = [1, 4, 5, 9, 10, 40, 50, 90,  
      100, 400, 500, 900, 1000]  
AA = ["I", "IV", "V", "IX", "X", "XL",  
      "L", "XC", "C", "CD", "D", "CM", "M"]  
i = 12  
KKK = int(input("DIGITE UN NÚMERO: "))  
while KKK:  
    EE = KKK // NN[i]  
    KKK %= NN[i]  
    while EE:  
        print(AA[i], end="")  
        EE -= 1  
    i -= 1
```

EXPLICA
EL
PROCESO

```
print("CAMBIO VALORES")  
DD = ["True", "False", "True", "False",  
      "False", "True", "True", "False",  
      "True", "True", "True", "False"]  
DD_NEW = []  
  
for i in DD:  
    if i == "True":  
        DD_NEW.append(1)  
    else:  
        DD_NEW.append(0)  
  
print(DD)  
print(DD_NEW)  
print(" ")  
DD_DOS = [1 if i == "True" else 0 for i in DD]  
print(DD)  
print(DD_DOS)
```

EXPLICA
EL
PROCESO

Vamos a crear un juego de adivinar el número que eligió la PC.

PASO 1: que la maquina elija un número entre 1 y 10.

PASO 2: el usuario intenta adivinar el número.

PASO 3: si no adivina, que se dé una oportunidad, diciéndole si el número de la PC fue más grande o más pequeño.

PASO 4: que esto se repita en tres oportunidades.



```
2 FF = ["Apple", "Bnana", "sfdsgdgd",  
3       "dgdfgdg", "dagdgdf", "dgdsgdga"]  
4 for FFs in FF:  
5     print(FFs)  
6  
7 print(" ")  
8 print("COMPRENSIÓN LISTAS")  
9 [print(i) for i in FF]
```



```
print(" ")  
print("VOLVER MAYUSCULAS")  
i = [i.upper() for i in FF]  
print(i)
```

Practiquemos:

PASO 1: Cree una lista para escribir el nombre de tus compañeros.

PASO 2: use un bucle **for** con **range** e **ingrese** datos a la lista vacía.

PASO 3: `print(Lista1, lista2)` cómo programador, ¿te satisface el resultado?

Nota: recuerda el uso del bucle **for**

```
for x in range(3):  
    print(TABLA1[x], TABLA2[x][0])
```

```
word = input("Introduce una palabra: ")  
reversed word = word  
word = list(word)  
reversed word = list(reversed word)  
reversed word.reverse()  
print(reversed word)
```

ADICIONAL 1: Investiga la sucesión de Fibonacci y crea un código para ello.

ADICIONAL 2: cree un programa que cuente las vocales de un texto incorporado por el usuario.

GUÍA DE APRENDIZAJE

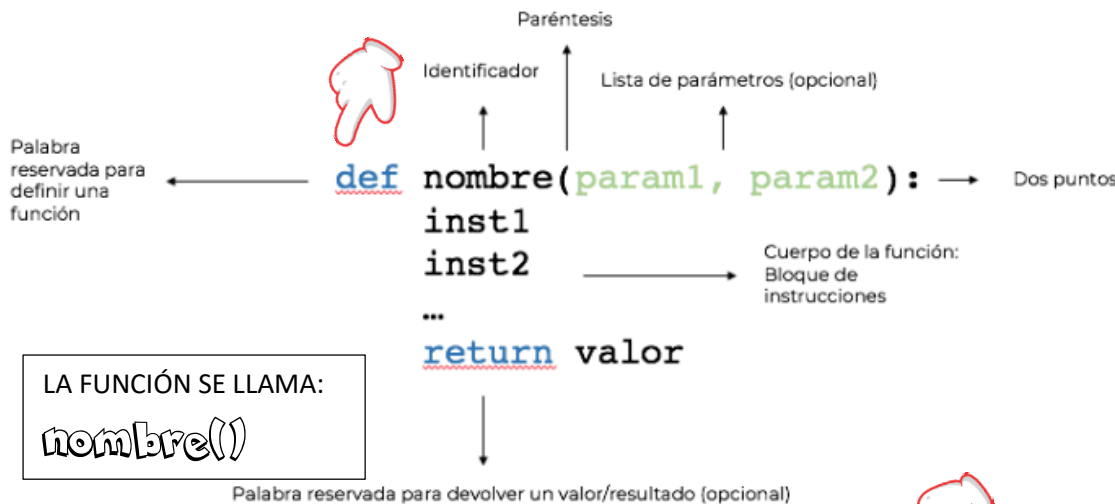


Python
Libraries

For GIS and Mapping

Una función es un bloque de código que solo se ejecuta cuando se le llama. Puede pasar datos, conocidos como parámetros, a una función. Una función puede devolver datos como resultado.

Ahora sí, pongámonos serios.



PYTHON tiene muchas funciones propias predefinidas, **INVESTIGALAS** y mantenlas presentes, ya que tarde que temprano **REQUERIRÁS** de su uso.

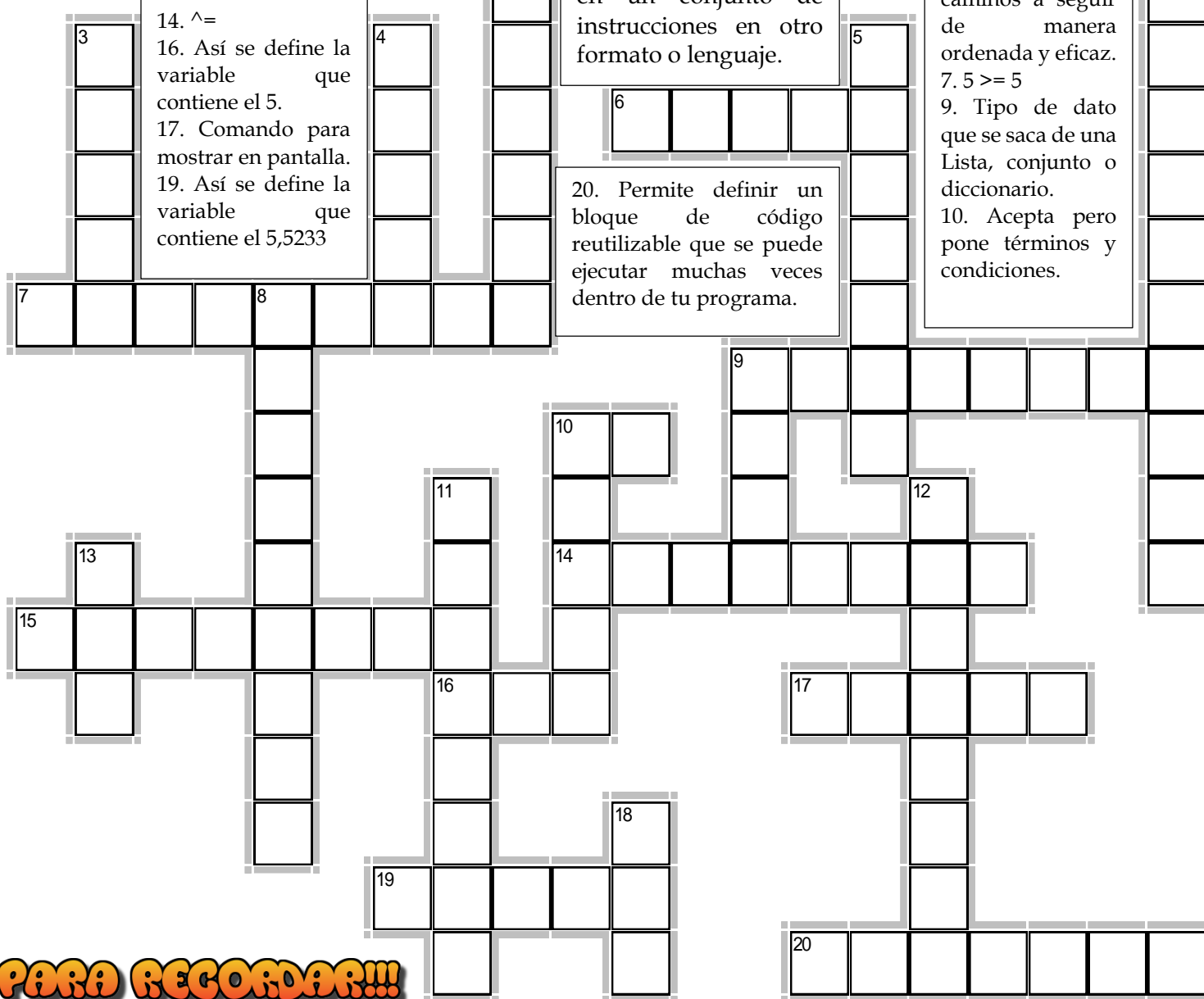
```
1 def HH():
2     print("")
3     password_1 = input("")
4     limite = 3
5     print(f"Tiene {limite} intentos")
6     password_2 = input("")
7     contador = 1
8
9     while password_1 != password_2 and contador < limite:
10        print("Inténtelo de nuevo.")
11        password_2 = input("")
12        contador += 1
13
14    if password_1 == password_2:
15        print("Contraseña confirmada.")
16    else:
17        print("No ha confirmado la contraseña.")
```

Analiza
esta
función y
complétala.

Un **SCRIPT** es un código particularmente creado para resolver una tarea específica. Normalmente cumple una o varias de las siguientes características:

- Se compone de un único módulo de python (un fichero de texto con extensión .py)
- Tiene un punto entrada/ejecución al final del mismo.
- Hace uso de librerías del sistema y de comandos de bash.
- Se lanza por consola.
- Se integra con facilidad con tareas tipo cron (cronjobs).
- Define diferentes funciones simples.
- Permite el uso de argumentos desde la consola.

GUÍA DE APRENDIZAJE



15. Proceso de transformar un programa informático escrito en un lenguaje en un conjunto de instrucciones en otro formato o lenguaje.

6. Denota los caminos a seguir de manera ordenada y eficaz.
7. $5 \geq 5$
9. Tipo de dato que se saca de una Lista, conjunto o diccionario.
10. Acepta pero pone términos y condiciones.

20. Permite definir un bloque de código reutilizable que se puede ejecutar muchas veces dentro de tu programa.

PARA RECORDAR!!!

3. Función que devuelve una secuencia de números, comenzando desde 0 de forma predeterminada, se incrementa en 1 y se detiene antes de un número específico.
4. Evalúa una condición y luego ejecuta un bloque de código si la condición es verdadera.
5. Así se define la variable que contiene el $5 + 3j$
8. Serie de comandos ordenados para realizar una tarea.
9. Python's Integrated Development and Learning Environment.
10. Comando para interactuar con el usuario.

11. Espacio virtual y transitorio, para guardar un tipo de dato.
12. $\% =$
13. Bucle que se utiliza para recorrer los elementos de un objeto iterable.
18. Es una variable que almacena cadena de caracteres.
1. Es un espacio de trabajo en el cuál todos los códigos tienen un esquema que los une.
2. Unir varios códigos en uno solo.

GUÍA DE APRENDIZAJE

Una función también se llama **método**.

```
1 def SS(num):  
2     x = num * 2  
3     return x  
4 print(x)  
5  
6 GG = input("bla bla ")  
7 print(SS(GG))
```

Error???

Al crear **funciones** en un **módulo principal**, se debe crear la función **main()** para que no genere error.

```
29 # FUNCIONES CON PARAMETROS  
30 """  
31 def miFuncion2(parametro1,parametro2):  
32     #conjunto de instrucciones  
33     """
```

EN ESTE EJEMPLO SE CREAN FUNCIONES Y LUEGO SE USAN CON OTRAS VARIABLES.

```
15 def derechos_humanos():  
16     print("Derecho a la vida")  
17     print("Derecho a la igualdad ante la ley")  
18     print("Derecho a la libertad")  
19  
20 derechos_humanos()  
21  
22 def derechos_mayorDeEdad():  
23     print("Derecho a votar")  
24     print("Derecho al trabajo")  
25  
26 def mayoria_de_edad(nombre,edad):  
27     print(f'Según la edad de {nombre}, sus derechos son:')  
28     if edad >= 18:  
29         derechos_humanos()  
30         derechos_mayorDeEdad()  
31     else:  
32         derechos_humanos()  
33  
34 def mayoria_de_edad2(edad,nombre='DESCONOCIDO'):  
35     print(f'Según la edad de {nombre}, sus derechos son:')  
36     if edad >= 18:  
37         derechos_humanos()  
38         derechos_mayorDeEdad()  
39     else:  
40         derechos_humanos()  
41  
42 MM = int(input("digite su edad"))  
43 SS = input("ESCRIBA SU NOMBRE")  
44 mayoria_de_edad2(MM)
```

CUÁNDO LLAME LA FUNCIÓN, PRIMERO ESCRÍBALA TAL CUAL, LUEGO ESCRIBA AMBAS VARIABLES.

```
1 class NAVAJASUIZA:  
2     def __init__(self):
```

Esta función que pasa a ser método por estar dentro de una clase, permite crear y llamar variables o métodos de la clase. Es con doble piso __

Clase: NAVAJASUIZA

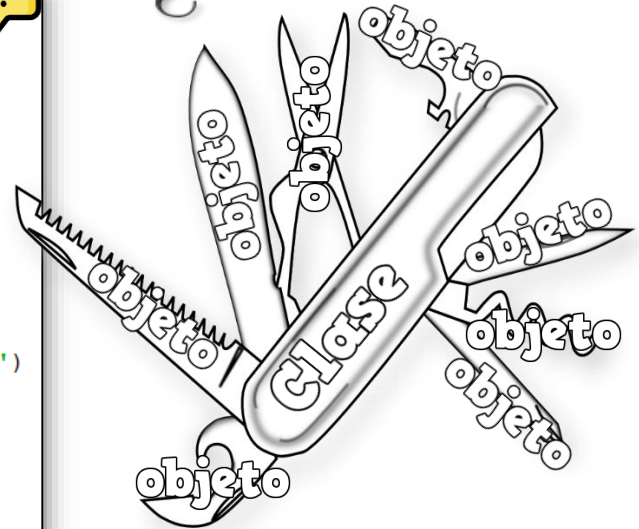
Objeto: Sierra

Atributo: dientes, plateada, 3,5 cm de largo.

Método: cortar desbastando.

```
1 def multiplica_por_5(numero):  
2     print(f'{numero} * 5 = {numero * 5}')  
3  
4 print('Comienzo del programa')  
5  
6 BB = int(input("BLA BLA"))  
7  
8 multiplica_por_5(BB)  
9  
10 print('Siguiendo')  
11 cc = int(input("BLA BLA"))  
12 multiplica_por_5(cc)  
13 print('Fin')
```

Clases



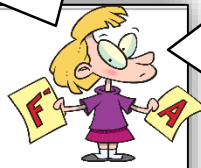
Crear un **objeto** a partir de una clase se denomina **instanciar**.

La **clase** es: **navaja inglesa** y cada uno de sus **objetos** tiene una **información** específica y una **acción** determinada.

Pero esta **navaja** solo es una **herramienta** entre otras muchas, para sobrevivir en un **ambiente específico**.

GUÍA DE APRENDIZAJE

ENCUENTRA EL ERROR Y
DEFINE QUE SE HIZO CON
ESTA CLASE.



```
1 class Maria:
2     def __init__(self, data):
3         self.data = data
4
5     def print_data(self):
6         print("EL DATO ES: ", self.data )
7
8 ff = input("BLA BLA: ")
9 p = maria(ff)
10 p.print_data()
11
```

Todo **MÉTODO** que contenga
doble piso __ (DUNDER)
es usado para un proceso **ESPECIAL** en
Python, por ejemplo **__int__**

A esta parte se le llama
CONSTRUCTOR.

```
def __init__(self, data):
    self.data = data
```

VAMOS A DAR ORDEN DE PROCESO.
ENCAPSULAMIENTO

```
1 def DD(KKK):
2     NN = [1, 4, 5, 9, 10, 40, 50, 90,
3           100, 400, 500, 900, 1000]
4     AA = ["I", "IV", "V", "IX", "X", "XL",
5           "M"]
6     i = 12
7
8     while KKK:
9         EE = KKK // NN[i]
10        KKK %= NN[i]
11        while EE:
12            print(AA[i], end="")
13            EE -= 1
14        i -= 1
15
16 KK = int(input(" "))
17 print(" ", end=" ")
18
```

COMPLÉTALA Y
EXPLICA LOS
CAMBIOS



```
1 import clases
2
3 def DD(KK):
4     print("HGHSFSDA")
5
6 def main():
7     clases.suma()
8     DD(KK)
9
10 if __name__ == "__main__":
11     main()
```

1. **Clases** está en otro módulo.
2. **DD** fue creada en el módulo principal.
3. **Main** es el cuerpo del programa.
4. **Main()** permite hacer funcionar el programa

```
def buscarCaracteres(ttt):
    limpio = ttt.replace(' ', '')
    lista de caracteres = list(limpio)

    caracteres = ",".join(lista de caracteres)
    print("QQQQ " + ttt + " QQQQ " + caracteres
          + "SON " + str(len(ttt)) + " letras.")

    sss = str(input("escriba su nombre: "))
    buscarCaracteres(sss)
```

El **módulo principal** lo puedes llamar como **desees**,
pero su función principal debe ser **main()**

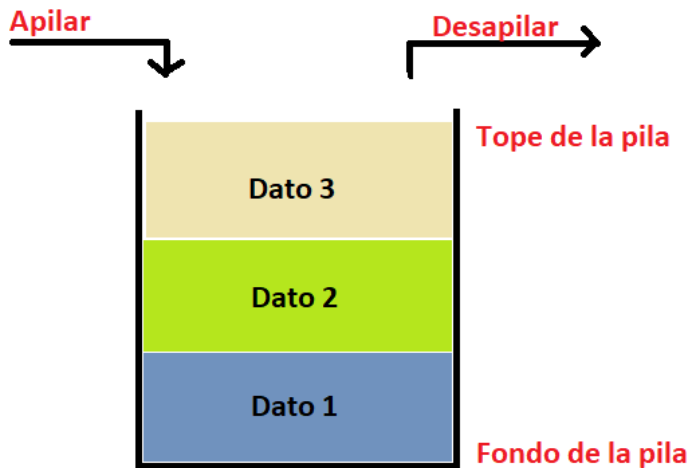
```
3 num_uno = 5
4
5 def DD(KKK):
6     global num_uno -= 1
7     print("lkjh")
8
9 def main():
10    global num_uno += 1
11    clases.suma()
12    DD()
```

Ahora tenemos
variables locales
(validas solo dentro de
la función) y
variables globales
(escritas una vez y
usadas en cualquier
función o método.

```
num_uno = 5

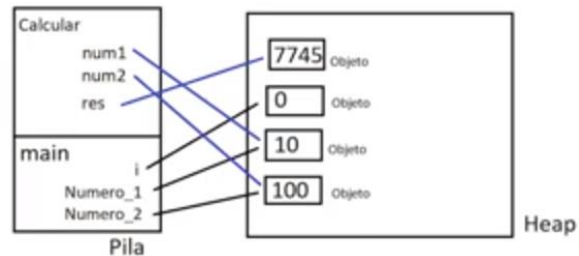
def DD(KKK):
    global num_uno -= 1
```


GUÍA DE APRENDIZAJE



Con este ejemplo puede definir qué es una **función recursiva**?

```
1 def factorial(num):
2     print("Valor inicial ->", num)
3     if num > 1:
4         num = num * factorial(num - 1)
5     print("valor final ->", num)
6     return num
7
8 NN = int(input("NUMERO INICIA: "))
9 print(factorial(NN))
```



Una **pila** consiste en una estructura en la que los **datos** a manejar se **almacenan** de forma ordenada, en una lista, de tal modo que puedan, en un determinado momento, ser recuperados por el programa, para trabajar sobre ellos, siendo el modo de acceso a dichos datos del tipo **LIFO** (iniciales en inglés de «**último en entrar, primero en salir**»), lo cual, ilustra el hecho de que en este tipo de estructura, el dato que se saca, será siempre el último que se haya introducido. El manejo de los datos implicados, se lleva a cabo a través de dos acciones principales:

El **apilamiento**, mediante el cual, se añaden nuevos elementos (o datos) a la pila.

El **desapilamiento**, mediante el cual se realiza la operación inversa consistente en sacar el último elemento añadido de la pila.

A su vez, hay que recordar que, en cada acción solo se podrá acceder al últimos elementos añadido (**ubicado en el tope de la pila**) de modo que su retirada, permitirá acceder al elementos anterior que pasará a ser el nuevo elemento ubicado en el tope.



Yo soy un programa de **Python**, los **módulos** son mi **caja de herramientas**, algunos los hago yo, pero otros ya vienen en **Python** y solo es llamarlos.

```
print(help("modules"))
```

Te mostrará los módulos disponibles en tu versión de **Python**.

```
2 import math
3 x = int(input("Ingresa un numero \n"))
4 y = math.sqrt(x)
5 print(y)
```

Línea 2: se importa el módulo **math**

Línea 4: se llama una función del módulo **math**

```
2 import math as mm
3 from math import sqrt
```

Línea 2: se importa el módulo **math** con nombre que yo quiera.

Línea 3: se llama solo la función **sqrt** del módulo **math**

GUÍA DE APRENDIZAJE

```
2 from math import sqrt
3 ss = int(input("hjhj: "))
4 print(sqrt(ss))
5
6 import math as mm
7 ss = int(input("hjhj: "))
8 print(mm.sqrt(ss))
```



Estos métodos solo llaman los módulos cuando están en la misma carpeta que el script `main()` o el `syspath` donde están las predeterminadas de Python.



En la página 7 hablamos de Paquetes, analiza que sería un subpaquete y mira la imagen.

```
2 from paquete.modulo import funcion
3 from paquete.sbp paquete.modulo import funcion
```



Una buena práctica es definir muy bien cada clase para que realice y una tarea específica, así si falla el programa, saber dónde ir a buscar y reparar.

Manejo de Datos



```
3 #CREAR ARCHIVO
4 variable_uno = open("blabla.txt", "w")
5 variable_dos = "blablabla \n blablabla"
6 variable_uno.write(variable_dos)
7 variable_uno.close()
8
9 #SOLO LECTURA
10 variable_uno = open("blabla.txt", "r")
11 print(variable_uno.readlines())
12 print(variable_uno.read())
13
14 #ESCRIBIR EN ARCHIVO
15 variable_uno = open("blabla.txt", "a")
16 variable_uno.write("\n blablabla")
17
```

Línea 4: crear .txt
Línea 5: contenido
Línea 6: incorporar contenido.

Write = w
Read = r
Append = a

