LABORATORIOS

OpenWebinars

LABORATORIO: PRINT()

- El comando **print()**, el cual es una de las funciones más sencillas de Python, simplemente imprime una línea de texto en la pantalla

LABORATORIO: PRINT()

- En tu primer laboratorio:
  - Utiliza la función **print()** para imprimir la línea ”¡Hola, Mundo!” en la pantalla
  - Una vez hecho esto, utiliza la función **print()** nuevamente, pero esta vez imprime tu nombre
  - Elimina las comillas dobles y ejecuta el código. Observa la reacción de Python, ¿qué tipo de error se produce?

LABORATORIO: PRINT()

- En tu primer laboratorio:
    - Luego, elimina los paréntesis, vuelve a poner las comillas dobles y vuelve a ejecutar el código. ¿Qué tipo de error se produce esta vez?
    - Experimenta tanto como puedas. Cambia las comillas dobles a comillas simples, utiliza múltiples funciones **print()** en la misma línea y luego en líneas diferentes. Observa que es lo que ocurre

LABORATORIO: PRINT() Y SUS ARGUMENTOS

- En el siguiente programa:

```
print(”Fundamentos”,”Programación”,”en”)
print(”Python”)
```

LABORATORIO: PRINT() Y SUS ARGUMENTOS

- Modifica la primera línea de código, utilizando las palabras claves **sep** y **end**, para que coincida con el resultado esperado
- Recuerda, utilizar dos funciones **print()**
- No cambies nada en la segunda invocación de **print()**

LABORATORIO: PRINT() Y SUS ARGUMENTOS

Salida Esperada

```
Fundamentos***Programación***en...Python
```

LABORATORIO: DANDO FORMA A LA SALIDA

```
print("    *")
print("   * *")
print("  *   *")
print(" *     *")
print("***   ***")
print("  *   *")
print("  *   *")
print("  *****")
```

LABORATORIO: DANDO FORMA A LA SALIDA

- Minimizar el número de invocaciones de la función **print()** insertando la secuencia **\n** en las cadenas
- Hacer la flecha dos veces más grande (pero mantener las proporciones)
- Duplicar la flecha, colocando ambas flechas lado a lado; Ten en cuenta que una cadena se puede multiplicar usando el siguiente truco: **"cadena" * 2** producirá **cadenacadena** (te contaremos más sobre ello pronto)

LABORATORIO: DANDO FORMA A LA SALIDA

- Elimina cualquiera de las comillas y observa detenidamente la respuesta de Python; presta atención a donde Python ve un error: ¿es el lugar en donde realmente existe el error?
- Haz lo mismo con algunos de los paréntesis

LABORATORIO: DANDO FORMA A LA SALIDA

- Cambia cualquiera de las palabras print en otra cosa (por ejemplo de minúscula a mayúscula, Print) - ¿Qué sucede ahora?
- Reemplaza algunas de las comillas por apóstrofes; observa lo que pasa detenidamente

LABORATORIO: LITERALES CADENAS

- Escribe una sola línea de código, utilizando la función **print()**, así como los caracteres de nueva línea y escape, para obtener la salida esperada de tres líneas

LABORATORIO: LITERALES CADENAS

Salida Esperada

```
”Estoy”
””aprendiendo””
”””Python”””
```

LABORATORIO: VARIABLES

- Erase una vez en la Tierra de las Manzanas, Juan tenía tres manzanas, María tenía cinco manzanas, y Adán tenía seis manzanas. Todos eran muy felices y vivieron por muchísimo tiempo. Fin de la Historia

LABORATORIO: VARIABLES

- Tu tarea es:
  - Crear las variables: **juan**, **maria**, y **adan**
  - Asignar valores a las variables. El valor debe de ser igual al número de manzanas que cada quien tenía
  - Una vez almacenados los números en las variables, imprimir las variables en una línea, y separar cada una de ellas con una coma

LABORATORIO: VARIABLES

- Tu tarea es:
    - Después se debe crear una nueva variable llamada **total_manzanas** y se debe igualar a la suma de las tres variables anteriores
    - Imprime el valor almacenado en **total_manzanas** en la consola

LABORATORIO: VARIABLES

- Tu tarea es:
  - Experimenta con tu código: crea nuevas variables, asigna diferentes valores a ellas, y realiza varias operaciones aritméticas con ellas (por ejemplo, **+**, **-**, *****, **/**, **//**, **etc**)
  - Intenta poner una cadena con un entero juntos en la misma línea, por ejemplo, **”Número Total de Manzanas:”** y **total_manzanas**

LABORATORIO: VARIABLES, UN CONVERTIDOR SIMPLE

- Millas y kilómetros son unidades de longitud o distancia
    - Teniendo en mente que 1 milla equivale aproximadamente a 1.61 kilómetros, complementa el siguiente programa para que convierta de:
        - Millas a kilómetros
        - Kilómetros a millas

LABORATORIO: VARIABLES, UN CONVERTIDOR SIMPLE

```
kilometers = 12.25

miles = 7.38

miles_to_kilometers = ###

kilometers_to_miles = ###

print(miles, ”millas son”, round(miles_to_kilometers, 2), ”kilómetros”)

print(kilometers, ”kilómetros son”, round(kilometers_to_miles, 2),
”millas”)
```

LABORATORIO: VARIABLES, UN CONVERTIDOR SIMPLE

- No se debe cambiar el código existente. Escribe tu código en los lugares indicados con **###.** Prueba tu programa con los datos que han sido provistos en el código fuente

- Pon mucha atención a lo que está ocurriendo dentro de la función **print()**. Analiza cómo es que se proveen múltiples argumentos para la función, y cómo es que se muestra el resultado

LABORATORIO: VARIABLES, UN CONVERTIDOR SIMPLE

- Nota que algunos de los argumentos dentro de la función **print()** son cadenas (por ejemplo **”millas son”**, y otros son variables (por ejemplo miles))

- Hay una cosa interesante más que está ocurriendo. ¿Puedes ver otra función dentro de la función **print()**? Es la función **round()**

LABORATORIO: VARIABLES, UN CONVERTIDOR SIMPLE

- Su trabajo es redondear la salida del resultado al número de decimales especificados en el paréntesis, y regresar un valor flotante (dentro de la función round() se puede encontrar el nombre de la variable, una coma, y el número de decimales que se desean mostrar)
- Se hablará más de esta función muy pronto, no te preocupes si no todo queda muy claro. Solo se quiere impulsar tu curiosidad

LABORATORIO: VARIABLES, UN CONVERTIDOR SIMPLE

- Después de realizar el programa, intenta escribir diferentes convertidores, por ejemplo, un convertidor de USD a EUR, un convertidor de temperatura, etc

LABORATORIO: VARIABLES, UN CONVERTIDOR SIMPLE

RESULTADO ESPERADO

7.38 millas son 11.88 kilómetros

12.25 kilómetros son 7.61 millas

LABORATORIO: OPERADORES Y EXPRESIONES

- La tarea es completar el código para poder calcular la siguiente expresión: $3x^3 - 2x^2 + 3x - 1$
- El resultado debe de ser asignado a y. Puedes utilizar variables adicionales para acortar la expresión (sin embargo, no es muy necesario)

LABORATORIO: OPERADORES Y EXPRESIONES

- Prueba tu código cuidadosamente:

```
x =  # codifica aquí tus datos de prueba
# escribe tu código aquí
print(”y =”, y)
```

LABORATORIO: OPERADORES Y EXPRESIONES

DATOS DE PRUEBA

Entrada de muestra: 0 | Salida esperada: y = -1

Entrada de muestra: 1 | Salida esperada: y = 3.0

Entrada de muestra: -1 | Salida esperada: y = -9

LABORATORIO: COMENTARIOS

- Añade comentarios para que el código sea más legible
- Cambia el nombre de las variables para que mejoren la claridad del código
- El programa calcula el número de segundos de las horas que tenemos guardada en la variable **h**
- En la variable **s** guardamos el total de segundos de una hora

LABORATORIO: COMENTARIOS

- Tenemos el siguiente programa:

```
h = 2
s = 3600
print(”Horas: ”, h)
print(”Segundos en Horas: ”, h * s)
```

LABORATORIO: ENTRADAS Y SALIDAS SIMPLES

```
# ingresa un valor flotante para la variable a aquí
# ingresa un valor flotante para la variable b aquí
# muestra el resultado de la suma aquí
# muestra el resultado de la resta aquí
# muestra el resultado de la multiplicación aquí
# muestra el resultado de la división aquí
print(”\n¡Eso es todo, amigos!”)
```

LABORATORIO: ENTRADAS Y SALIDAS SIMPLES

- La tarea es completar el código para evaluar y mostrar el resultado de cuatro operaciones aritméticas básicas
- El resultado debe ser mostrado en consola
- Quizá no podrás proteger el código de un usuario que intente dividir entre cero
    - Por ahora, no hay que preocuparse por ello

LABORATORIO: OPERADORES Y EXPRESIONES

- La tarea es completar el código para poder evaluar la siguiente expresión:
    - El resultado debe de ser asignado a y. Se cauteloso, observa los operadores y priorízalos. Utiliza cuantos paréntesis sean necesarios

$$\cfrac{1}{x+\cfrac{1}{x+\cfrac{1}{x+\cfrac{1}{x}}}}$$

LABORATORIO: OPERADORES Y EXPRESIONES

- Puedes utilizar variables adicionales para acortar la expresión (sin embargo, no es muy necesario)
- Prueba tu código cuidadosamente:

```
x = float(input("Ingresa el valor para x: "))
# Escribe tu código aquí.
print("y =", y)
```

LABORATORIO: OPERADORES Y EXPRESIONES

DATOS DE PRUEBA

Entrada de muestra: 1 | Salida esperada: y = 0.6000000000000001

Entrada de muestra: 10 | Salida esperada: y = 0.09901951266867294

Entrada de muestra: 100 | Salida esperada: y = 0.009999000199950014

Entrada de muestra: -5 | Salida esperada: y = -0.19258202567760344

LABORATORIO: CÁLCULO DE HORAS

- La tarea es preparar un código simple para calcular el tiempo final de un periodo de tiempo dado, expresándolo en horas y minutos. Las horas van de 0 a 23 y los minutos de 0 a 59. El resultado debe ser mostrado en la consola.

- Por ejemplo, si el evento comienza a las 12:17 y dura 59 minutos, terminará a las 13:16

LABORATORIO: CÁLCULO DE HORAS

- Pista: utilizar el operador % puede ser clave para el éxito

```
hour = int(input("Hora de inicio (horas): "))
mins = int(input("Minuto de inicio (minutos): "))
dura = int(input("Duración del evento (minutos): "))
# Escribe tu código aquí
```

LABORATORIO: CÁLCULO DE HORAS

ENTRADA DE MUESTRA

```
12
17
59
```

LABORATORIO: CÁLCULO DE HORAS

SALIDA ESPERADA

```
13:16
```

LABORATORIO: ESTRUCTURA SECUENCIAL

- **Ejercicio 1**
  - Escribir un programa que convierta un valor dado en grados Fahrenheit a grados Celsius.
  - Recordar que la fórmula para la conversión es:
    - **C = (F-32)*5/9**

LABORATORIO: ESTRUCTURA SECUENCIAL

- **Ejercicio 2**
  - Realiza un programa que reciba una cantidad de minutos y muestre por pantalla a cuantas horas y minutos corresponde
  - Por ejemplo: **1000 minutos** son **16 horas y 40 minutos**

LABORATORIO: ESTRUCTURA SECUENCIAL

- **Ejercicio 3**
  - Un alumno desea saber cuál será su calificación final en la materia de Algoritmos. Dicha calificación se compone de los siguientes porcentajes:
  - 55% del promedio de sus tres calificaciones parciales
  - 30% de la calificación del examen final
  - 15% de la calificación de un trabajo final

LABORATORIO: PREGUNTAS Y RESPUESTAS

- Usando uno de los operadores de comparación en Python, escribe un programa simple, de dos líneas que tome el parámetro **n** como entrada, que es un entero, e imprime **False** si **n** es menor que 100, y **True** si **n** es mayor o igual que 100
- No debes crear ningún bloque **if** (hablaremos de ellos muy pronto)
- Prueba tu código usando los datos que te proporcionamos

LABORATORIO: PREGUNTAS Y RESPUESTAS

DATOS DE PRUEBA

Ejemplo de entrada: 55 | Resultado esperado: False

Ejemplo de entrada: 99 | Resultado esperado: False

Ejemplo de entrada: 100 | Resultado esperado: True

Ejemplo de entrada: 101 | Resultado esperado: True

Ejemplo de entrada: -5 | Resultado esperado: False

Ejemplo de entrada: +123 | Resultado esperado: True

LABORATORIO: EJECUCIÓN CONDICIONAL

- Escribe un programa que utilice el concepto de ejecución condicional, tome una cadena como entrada y que:
    - Imprima el enunciado ”Si, PYTHON es el mejor lenguaje de programación” en la pantalla si la cadena ingresada es ”PYTHON”
    - Imprima ”No, me gusta más PYTHON” si la cadena ingresada es ”python”
    - Imprima ”¡PYTHON!, ¡No [entrada]!” de lo contrario. [entrada] es la cadena que se toma como entrada

LABORATORIO: EJECUCIÓN CONDICIONAL

- Prueba tu código con los datos que te proporcionamos. ¡Y hazte de un PYTHON también!

LABORATORIO: PREGUNTAS Y RESPUESTAS

DATOS DE PRUEBA

Entrada de muestra: python

Resultado esperado: No, me gusta más PYTHON

Entrada de ejemplo: Java

Resultado esperado: ¡PYTHON!, ¡No Java!

Entrada de muestra: PYTHON

Resultado esperado: Si, PYTHON es el mejor lenguaje de programación

LABORATORIO: IF - ELSE

- Queremos crear un programa para que calcule el Impuesto Personal de Ingresos (IPI, para abreviar) que se calcula utilizando la siguiente regla:
    - Si el ingreso del ciudadano no es superior a 85.528 euros, el impuesto es igual al 18% del ingreso menos 556 euros y 2 céntimos (exención fiscal)
    - Si el ingreso es superior a esta cantidad, el impuesto es igual a 14.839 euros y 2 céntimos, más el 32% del excedente sobre 85.528 euros

LABORATORIO: IF - ELSE

- Tu tarea es escribir una calculadora de impuestos
  - Debe aceptar un valor de punto flotante: el ingreso
  - A continuación, debe imprimir el impuesto calculado, redondeado a euros totales

LABORATORIO: IF - ELSE

- Hay una función llamada **round()** que hará el redondeo por ti, la encontrarás en el código que puedes usar de plantilla:
  - Si el impuesto calculado es menor que cero, solo significa que no hay impuesto (el impuesto es igual a cero). Ten esto en cuenta durante tus cálculos

LABORATORIO: IF - ELSE

```
income = float(input("Introduce el ingreso anual:"))

# Escribe tu código aquí.

tax = round(tax, 0)

print("El impuesto es:", tax, "euros")
```

LABORATORIO: IF - ELIF - ELSE

- Un año puede ser bisiesto o común. Los primeros tienen una duración de 366 días, mientras que los últimos tienen una duración de 365 días

LABORATORIO: IF - ELIF - ELSE

- Desde la introducción del calendario Gregoriano (en 1582), se utiliza la siguiente regla para determinar el tipo de año:
    - Si el número del año no es divisible entre cuatro, es un año común
    - De lo contrario, si el número del año no es divisible entre 100, es un año bisiesto
    - De lo contrario, si el número del año no es divisible entre 400, es un año común
    - De lo contrario, es un año bisiesto

LABORATORIO: IF - ELIF - ELSE

- Observa el código en el editor: solo lee un número de año y debe completarse con las instrucciones que implementan la prueba que acabamos de describir:

```
year = int(input("Introduce un año:"))

# Escribe tu código aquí
```

LABORATORIO: IF - ELIF - ELSE

- El código debe mostrar uno de los dos mensajes posibles, que son Año Bisiesto o Año Común, según el valor ingresado
- Sería bueno verificar si el año ingresado cae en la era Gregoriana y emitir una advertencia de lo contrario: No dentro del período del calendario Gregoriano
    - Consejo: utiliza los operadores **!=** y **%**

LABORATORIO: ESTRUCTURA ALTERNATIVA

- **Ejercicio 1**
    - Crea un programa que pida al usuario dos números y muestre su división si el segundo no es cero, o un mensaje de aviso en caso contrario

LABORATORIO: ESTRUCTURA ALTERNATIVA

- **Ejercicio 2**
    - Realiza un algoritmo que calcule la potencia, para ello pide por teclado la base y el exponente

LABORATORIO: ESTRUCTURA ALTERNATIVA

- Pueden ocurrir tres cosas:
  - El exponente sea positivo, sólo tienes que imprimir la potencia
  - El exponente sea 0, el resultado es 1
  - El exponente sea negativo, el resultado es 1/potencia con el exponente positivo

LABORATORIO: ESTRUCTURA ALTERNATIVA

- **Ejercicio 3**
  - Escribe un programa que pida una fecha (día, mes y año) y diga si es correcta

LABORATORIO: ADIVINA EL NÚMERO SECRETO

- Un mago junior ha elegido un número secreto. Lo ha escondido en una variable llamada **numero_secreto**
- Quiere que todos los que ejecutan su programa jueguen el juego Adivina el número secreto, y adivina qué número ha elegido para ellos
- ¡Quiénes no adivinen el número quedarán atrapados en un bucle sin fin para siempre! Desafortunadamente, él no sabe cómo completar el código

LABORATORIO: ADIVINA EL NÚMERO SECRETO

```
numero_secreto = 777
print(
"""
+=====================================+
| ¡Bienvenido a mi juego, muggle!     |
| Introduce un número entero          |
| y adivina qué número he             |
| elegido para ti.                    |
| Entonces,                           |
| ¿Cuál es el número secreto?         |
+=====================================+
""")
```

LABORATORIO: ADIVINA EL NÚMERO SECRETO

- Realiza un programa:
  - Pedirá al usuario que ingrese un número entero
  - Utilizará un bucle while
  - Comprobará si el número ingresado por el usuario es el mismo que el número escogido por el mago

LABORATORIO: ADIVINA EL NÚMERO SECRETO

- Información adicional: Observa la función **print()**
- La forma en que lo hemos utilizado aquí se llama impresión multilínea
- Puede utilizar comillas triples para imprimir cadenas en varias líneas para facilitar la lectura del texto o crear un diseño especial basado en texto
- Experimenta con ello

LABORATORIO: ADIVINA EL NÚMERO SECRETO

- Si el número elegido por el usuario es diferente al número secreto del mago, el usuario debería ver el mensaje **”¡Ja, ja! ¡Estás atrapado en mi bucle!”** y se le solicitará que ingrese un número nuevamente

LABORATORIO: ADIVINA EL NÚMERO SECRETO

- Si el número ingresado por el usuario coincide con el número escogido por el mago, el número debe imprimirse en la pantalla, y el mago debe decir las siguientes palabras: **”¡Bien hecho, muggle! Eres libre ahora”**.

LABORATORIO: ADIVINA EL NÚMERO SECRETO

- Información adicional: Observa la función **print()**.
- La forma en que lo hemos utilizado aquí se llama impresión multilínea
- Puede utilizar comillas triples para imprimir cadenas en varias líneas

LABORATORIO: BUCLE FOR: CONTADOR

- La idea detrás de esto es que agregar la palabra Mississippi a un número al contar los segundos en voz alta hace que suene más cercano al reloj, y por lo tanto ”uno Mississippi, dos Mississippi, tres Mississippi” tomará aproximadamente unos tres segundos reales de tiempo

LABORATORIO: BUCLE FOR: CONTADOR

- Tu tarea es muy simple aquí: escribe un programa que use un bucle for para ”contar de forma mississippi” hasta cinco. Habiendo contado hasta cinco, el programa debería imprimir en la pantalla el mensaje final **”¡Listos o no, ahí voy!”**

LABORATORIO: BUCLE FOR: CONTADOR

```
import time

    # Escribe un bucle for que cuente hasta cinco.

    # Cuerpo del bucle: imprime el número de iteración del bucle y la palabra ”Mississippi”.

    # Cuerpo del bucle - usar: time.sleep (1)

# Escribe una función de impresión con el mensaje final
```

LABORATORIO: BUCLE FOR: CONTADOR

- En el código anterior hay dos elementos que todavía no hemos explicado: la instrucción **import time** y el método **sleep()**
- Hemos importado el módulo time y hemos utilizado el método **sleep()** para suspender la ejecución de cada función posterior de **print()** dentro del bucle for durante un segundo, de modo que el mensaje enviado a la consola se parezca a un conteo real. No te preocupes, pronto aprenderás más sobre módulos y métodos

LABORATORIO: BREAK

- Diseña un programa que use un bucle while y le pida continuamente al usuario que ingrese una palabra a menos que ingrese "**chupacabra**" como la palabra de salida secreta, en cuyo caso el mensaje "**¡Has dejado el bucle con éxito**" debe imprimirse en la pantalla y el bucle debe terminar

LABORATORIO: BREAK

- Vamos a construir dos versiones del programa:
    - Versión 1: Utiliza el concepto de ejecución condicional y la instrucción **break**. En este caso el bucle no evaluará ninguna condición, es decir, será un bucle infinito.
    - Versión 2: Realmente no es necesario usar la instrucción **break**. Diseña una solución donde no se use **break** y el bucle while controle la condición de salida

LABORATORIO: CONTINUE

- ¡Debes diseñar un devorador de vocales! Escribe un programa que use:
  - Un bucle for
  - El concepto de ejecución condicional (if-elif-else)
  - La instrucción continue

LABORATORIO: CONTINUE

- Tu programa debe:
  - Pedir al usuario que ingrese una palabra
  - Utiliza **palabra_usuario = palabra_usuario.upper()** para convertir la palabra ingresada por el usuario a mayúsculas; hablaremos sobre los llamados métodos de cadena y el método **upper()** muy pronto, no te preocupes

LABORATORIO: CONTINUE

- Tu programa debe:
  - Utiliza la ejecución condicional y la instrucción continue para ”comer” las vocales A , E , I , O , U de la palabra ingresada
  - Imprime las letras no consumidas en la pantalla, cada una de ellas en una línea separada

LABORATORIO: CONTINUE

- **Plantilla:**

```
# Indicar al usuario que ingrese una palabra

# y asignarlo a la variable palabra_usuario

for letra in palabra_usuario:

    # Completa el cuerpo del bucle for
```

LABORATORIO: CONTINUE

DATOS DE PRUEBA

Entrada de muestra:

```
Gregory
```

Salida esperada:

```
G
R
G
R
Y
```

LABORATORIO: CONTINUE

- **Mejora**
  - Vamos a mejorar el Devorador de vocales
  - Modifica el programa anterior para que no vaya escribiendo cada letra, sino que las asigne a la variable **palabra_sin_vocales** e imprime la variable en la pantalla

LABORATORIO: CONTINUE

- Plantilla:

```
palabra_sin_vocales = ””

# Indicar al usuario que ingrese una palabra

# y asignarla a la variable user_word.

for letter in user_word:

  # Completa el cuerpo del bucle.

# Imprimir la palabra asignada a palabra_sin_vocales
```

LABORATORIO: CONTINUE

- Hemos creado **palabra_sin_vocales** y le hemos asignado una cadena vacía. Utiliza la operación de concatenación para pedirle a Python que añada las letras seleccionadas a dicha variable durante los siguientes giros de bucle

LABORATORIO: CONTINUE

DATOS DE PRUEBA

Entrada de muestra:

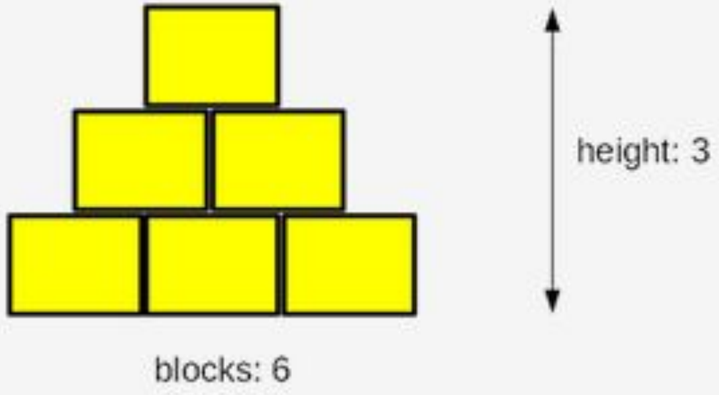
```
Gregory
```

Salida esperada:

```
GRGRY
```

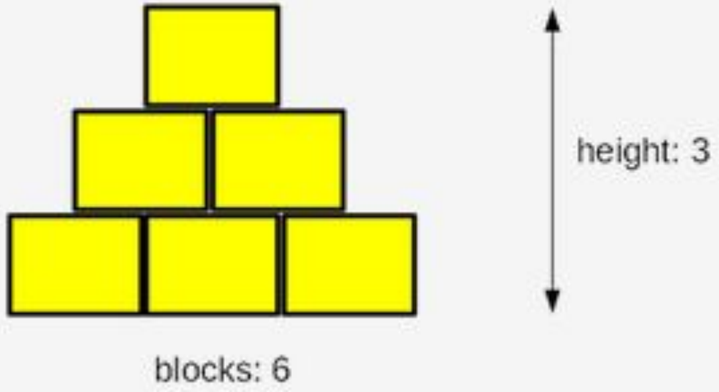
LABORATORIO: BUCLE WHILE

- Escucha esta historia: Un niño y su padre, un programador de ordenadores, juegan con bloques de madera
- Están construyendo una pirámide



LABORATORIO: BUCLE WHILE

- Su pirámide es un poco rara, ya que en realidad es una pared en forma de pirámide, es plana
- La pirámide se apila de acuerdo con un principio simple: cada capa inferior contiene un bloque más que la capa superior



LABORATORIO: BUCLE WHILE

- Tu tarea es escribir un programa que lea la cantidad de bloques que tienen los constructores, y generar la altura de la pirámide que se puede construir utilizando estos bloques
- La altura se mide por el número de capas completas: si los constructores no tienen la cantidad suficiente de bloques y no pueden completar la siguiente capa, terminan su trabajo inmediatamente

LABORATORIO: BUCLE WHILE

```
altura = 0

bloques = int(input("Ingresa el número de bloques: "))

#

# Escribe tu código aquí.

#

print("La altura de la pirámide:", altura)
```

LABORATORIO: BUCLE WHILE

DATOS DE PRUEBA

Entrada de muestra: 6

Salida esperada: La altura de la pirámide es: 3

Entrada de muestra: 20

Salida esperada: La altura de la pirámide es: 5

Entrada de muestra: 1000

Salida esperada: La altura de la pirámide es: 44

LABORATORIO: LA HIPÓTESIS DE COLLATZ

- En 1937, un matemático alemán llamado Lothar Collatz formuló una hipótesis intrigante (aún no se ha comprobado) que se puede describir de la siguiente manera:
    - Toma cualquier número entero que no sea negativo y que no sea cero y asígnale el nombre **c0**

LABORATORIO: LA HIPÓTESIS DE COLLATZ

- Si es par, evalúa un nuevo **c0** como **c0 / 2**
- De lo contrario, si es impar, evalúe un nuevo **c0** como **3 * c0 + 1**
- Si **c0** es distinto a 1, salta al punto 2
- La hipótesis dice que, independientemente del valor inicial de **c0**, el valor siempre tiende a 1

LABORATORIO: LA HIPÓTESIS DE COLLATZ

- Escribe un programa que lea un número natural y ejecute los pasos anteriores siempre que **c0** sea diferente de 1
- También queremos que cuente los pasos necesarios para lograr el objetivo
- Tu código también debe mostrar todos los valores intermedios de **c0**

LABORATORIO: LA HIPÓTESIS DE COLLATZ

DATOS DE PRUEBA

Entrada de muestra:

```
16
```

Salida esperada:

```
8
4
2
1
pasos = 4
```

LABORATORIO: ESTRUCTURA REPETITIVA

- **Ejercicio 1**
  - Realizar una algoritmo que muestre la tabla de multiplicar de un número introducido por teclado
  - Modifica el programa anterior para que no pida el número por teclado, sino que muestre las tablas de multiplicar de los 5 primeros números

LABORATORIO: ESTRUCTURA REPETITIVA

- **Ejercicio 2**
  - Algoritmo que pida números hasta que se introduzca un cero. Debe imprimir la suma y la media de todos los números introducidos
  - Realiza dos versiones: una que utiliza la instrucción **break** y otra no

LABORATORIO: ESTRUCTURA REPETITIVA

- **Ejercicio 3**
  - Realizar un algoritmo que pida números (se pedirá por teclado la cantidad de números a introducir)
  - El programa debe informar de cuantos números introducidos son mayores que 0, menores que 0 e iguales a 0

LABORATORIO: FUNDAMENTOS DE LISTAS

Queremos crear una lista de cinco números: **1, 2, 3, 4 y 5**

- Escribir una línea de código que solicite al usuario que reemplace el número central en la lista con un número entero ingresado por el usuario (Paso 1)
- Escribir una línea de código que elimine el último elemento de la lista (Paso 2)
- Escribir una línea de código que imprima la longitud de la lista existente (Paso 3)

LABORATORIO: FUNDAMENTOS DE LISTAS

```
lista = [1, 2, 3, 4, 5]

# Paso 1: reemplazar el número de en medio con un número entero ingresado por el usuario.

# Paso 2: escribe aquí una línea de código que elimine el último elemento de la lista

# Paso 3: escribe aquí una línea de código que imprima la longitud de la lista existente

print(lista)
```

LABORATORIO: LOS BEATLES

- Los Beatles fueron uno de los grupos de música más populares de la década de 1960 y la banda más vendida en la historia
- La banda sufrió muchos cambios de formación, que culminaron en 1962 con la formación de John Lennon, Paul McCartney, George Harrison y Richard Starkey (mejor conocido como Ringo Starr)

LABORATORIO: LOS BEATLES

1. Crea una lista vacía llamada beatles.
2. Emplea el método append() para agregar los siguientes miembros de la banda a la lista: John Lennon, Paul McCartney y George Harrison
3. Emplea el bucle for y el append() para pedirle al usuario que agregue los siguientes miembros de la banda a la lista: Stu Sutcliffe, y Pete Best

LABORATORIO: LOS BEATLES

4. Usa la instrucción del para eliminar a Stu Sutcliffe y Pete Best de la lista

5. Usa el método insert() para agregar a Ringo Starr al principio de la lista

LABORATORIO: LOS BEATLES

```
# paso 1
print(”Paso 1:”, Beatles)
# paso 2
print(”Paso 2:”, Beatles)
# paso 3
print(”Paso 3:”, Beatles)
# paso 4
print(”Paso 4:”, Beatles)
# paso 5
print(”Paso 5:”, Beatles)
# probando la longitud de la lista
print(”Número de Beatles:”, len(Beatles))
```

LABORATORIO: OPERACIONES CON LISTAS

- Tenemos una lista con números enteros. Algunos de estos números pueden estar repetidos, pero no queremos ninguna repetición
- **Queremos que las repeticiones sean eliminadas**
- **Escribe un programa que elimine todas las repeticiones de números de la lista**

LABORATORIO: OPERACIONES CON LISTAS

- Asume que la lista original está ya dentro del código, no tienes que introducirla desde el teclado
- Puedes mejorar el código y agregar una parte que pueda llevar a cabo una conversación con el usuario y obtener todos los datos
- **Sugerencia:** Te recomendamos que crees una nueva lista como área de trabajo temporal, no necesitas actualizar la lista actual

LABORATORIO: OPERACIONES CON LISTAS

```
my_list = [1, 2, 4, 4, 1, 4, 2, 6, 2, 9]

#

# Escribe tu código aquí.

#

print("La lista con elementos únicos:")

print(my_list)
```

LABORATORIO: EJERCICIOS CON LISTAS

- **Ejercicio 1**
    - Se quiere realizar un programa que lea por teclado las 5 notas obtenidas por un alumno (comprendidas entre 0 y 10). A continuación debe mostrar todas las notas, la nota media, la nota más alta que ha sacado y la menor

LABORATORIO: EJERCICIOS CON LISTAS

- **Ejercicio 2**
  - Diseñar el algoritmo correspondiente a un programa, que:
    - Crea una tabla (lista con dos dimensiones) de 5x5 enteros
    - Carga la tabla con valores numéricos enteros
    - Suma todos los elementos de cada fila y todos los elementos de cada columna visualizando los resultados en pantalla

LABORATORIO: EJERCICIOS CON LISTAS

- **Ejercicio 3**
  - Queremos guardar la temperatura mínima y máxima de 5 días. Realiza un programa que de la siguiente información:
    - La temperatura media de cada día
    - Los días con menos temperatura
    - Se lee una temperatura por teclado y se muestran los días cuya temperatura máxima coincide con ella. Si no existe ningún día se muestra un mensaje de información

LABORATORIO: AÑO BISIESTO

Tu tarea es escribir y probar una función que toma un argumento (un año) y devuelve **True** si el año es un año bisiesto, o **False** si no lo es.

- Si el número del año no es divisible entre cuatro, es un año común
- De lo contrario, si el número del año no es divisible entre 100, es un año bisiesto
- De lo contrario, si el número del año no es divisible entre 400, es un año común
- De lo contrario, es un año bisiesto

LABORATORIO: AÑO BISIESTO

```
def año_bisiesto(año):
#
# Escribe tu código aquí.
#
test_data = [1900, 2000, 2016, 1987]
test_results = [False, True, True, False]
for i in range(len(test_data)):
  yr = test_data[i]
  print(yr,"->",end="")
  result = año_bisiesto(yr)
  if result == test_results[i]:
    print("OK")
  else:
    print("Fallido")
```

LABORATORIO: CUÁNTOS DÍAS

- Tu tarea es escribir y probar una función que toma dos argumentos (un año y un mes) y devuelve el número de días del mes/año dado (mientras que solo febrero es sensible al valor año, tendrá 28 o 29 según el tipo de año)
- Haz que la función devuelva **None** si los argumentos no tienen sentido

LABORATORIO: CUÁNTOS DÍAS

```
def año_bisiesto(año):
#
# Tu código del laboratorio anterior
#


def dias_del_mes(año, mes):
#
# Escribe tu código aquí.
#
```

LABORATORIO: CUÁNTOS DÍAS

```
test_years = [1900, 2000, 2016, 1987]
test_months = [2, 2, 1, 11]
test_results = [28, 29, 31, 30]
for i in range(len(test_years)):
    yr = test_years[i]
    mo = test_months[i]
    print(yr, mo, "->", end="")
    result = dias_del_mes(yr, mo)
    if result == test_results[i]:
        print("OK")
    else:
        print("Fallido")
```

LABORATORIO: DÍA DEL AÑO

- Tu tarea es escribir y probar una función que toma tres argumentos (un año, un mes y un día del mes) y devuelve el día correspondiente del año, o devuelve **None** si cualquiera de los argumentos no es válido
- Debes utilizar las funciones previamente escritas y probadas. Agrega algunos casos de prueba al código

LABORATORIO: DÍA DEL AÑO

```
def año_bisiesto(año):
#
# Tu código del laboratorio anterior
#
def dias_del_mes(año, mes):
#
# Tu código del laboratorio anterior
#
def dia_del_año(año, mes, dia):
#
# Escribe tu código nuevo aquí.
#
print(dia_del_año(2000, 12, 31))
```

LABORATORIO: NÚMEROS PRIMOS

Un número natural es primo si es mayor que 1 y no tiene divisores más que 1 y sÍ mismo. Ejemplos:

- 8 no es un número primo, ya que puedes dividirlo entre 2 y 4.
- 7 es un número primo, ya que no podemos encontrar ningún divisor para él, sólo el 1 y el mismo.

LABORATORIO: NÚMEROS PRIMOS

Escribir una función que verifique si un número es primo o no.

- Se llama es_primo.
- Toma un argumento (el valor a verificar).
- Devuelve True si el argumento es un número primo, y False de lo contrario.

LABORATORIO: NÚMEROS PRIMOS

- Sugerencia: intenta dividir el argumento por todos los valores posteriores (comenzando desde 2) y verifica el resto: si es cero, tu número no puede ser un número primo; analiza cuidadosamente cuándo deberías detener el proceso.

LABORATORIO: NÚMEROS PRIMOS

```
def es_primo(num):
#
# Escribe tu código aquí.
#


for i in range(1, 20):
    if es_primo(i + 1):
            print(i + 1, end=" ")
print()
```

LABORATORIO: CONVERSIÓN DEL CONSUMO DE COMBUSTIBLE

- El consumo de combustible de un automóvil se puede expresar de muchas maneras diferentes. Por ejemplo:
  - En Europa, se muestra como la cantidad de combustible consumido por cada 100 kilómetros
  - En los EE. UU., se muestra como la cantidad de millas recorridas por un automóvil con un galón de combustible

LABORATORIO: CONVERSIÓN DEL CONSUMO DE COMBUSTIBLE

- Tu tarea es escribir un par de funciones que conviertan l/100km a mpg (millas por galón), y viceversa. Las funciones:
    - Se llaman **litros_100km_a_millas_galon** y **millas_galon_a_litros_100km** respectivamente
    - Tiene un parámetro con el valor a convertir

LABORATORIO: CONVERSIÓN DEL CONSUMO DE COMBUSTIBLE

- Aquí hay información para ayudarte:
  - 1 milla = 1609.344 metros
  - 1 galón = 3.785411784 litros

LABORATORIO: CONVERSIÓN DEL CONSUMO DE COMBUSTIBLE

```
def litros_100km_a_millas_galon(liters):
# Escribe tu código aquí.
def millas_galon_a_litros_100km(miles):
# Escribe tu código aquí.
print(litros_100km_a_millas_galon(3.9))
print(litros_100km_a_millas_galon(7.5))
print(litros_100km_a_millas_galon(10.))
print(millas_galon_a_litros_100km(60.3))
print(millas_galon_a_litros_100km(31.4))
print(millas_galon_a_litros_100km(23.5))
```

LABORATORIO: EJERCICIOS CON DICCIONARIOS

- **Ejercicio 1**
    - Escribe un programa python que pida un número por teclado y que cree un diccionario cuyas claves sean desde el número 1 hasta el número indicado, y los valores sean los cuadrados de las claves.

LABORATORIO: EJERCICIOS CON DICCIONARIOS

- **Ejercicio 2**
  - Vamos a crear un programa en python donde vamos a declarar un diccionario para guardar los precios de las distintas frutas. El programa pedirá el nombre de la fruta y la cantidad que se ha vendido y nos mostrará el precio final de la fruta a partir de los datos guardados en el diccionario. Si la fruta no existe nos dará un error. Tras cada consulta el programa nos preguntará si queremos hacer otra consulta.
  - Puedes usar la construcción cadena.lower() para convertir en minúscula la cadena guardada en la variable cadena.

LABORATORIO: EJERCICIOS CON DICCIONARIOS

- **Ejercicio 3**
    - Escribir un programa que implemente una agenda. En la agenda se podrán guardar nombres y números de teléfono. El programa nos dará el siguiente menú:

LABORATORIO: EJERCICIOS CON DICCIONARIOS

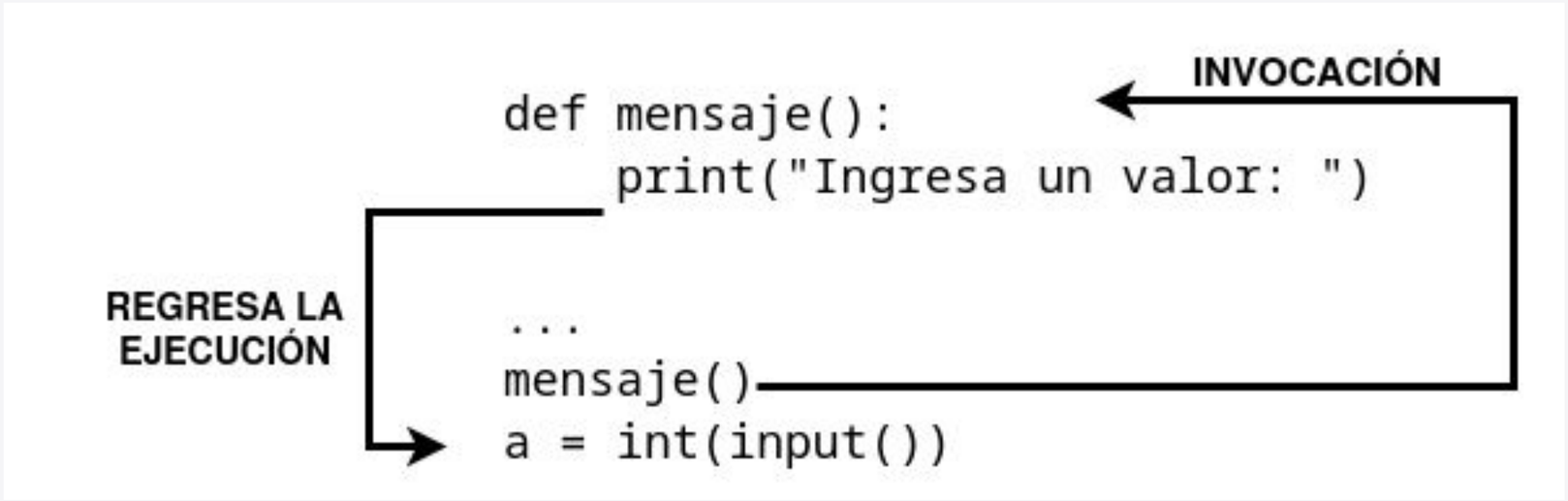
- **Ejercicio 3**
    - **Añadir/modificar**: Nos pide un nombre. Si el nombre se encuentra en la agenda, debe mostrar el teléfono y, opcionalmente, permitir modificarlo si no es correcto. Si el nombre no se encuentra, debe permitir ingresar el teléfono correspondiente.
    - **Buscar**: Nos pide una cadena de caracteres, y nos muestras todos los contactos cuyos nombres comiencen por dicha cadena. Podemos usar cadena.startswith(subcad).

LABORATORIO: EJERCICIOS CON DICCIONARIOS

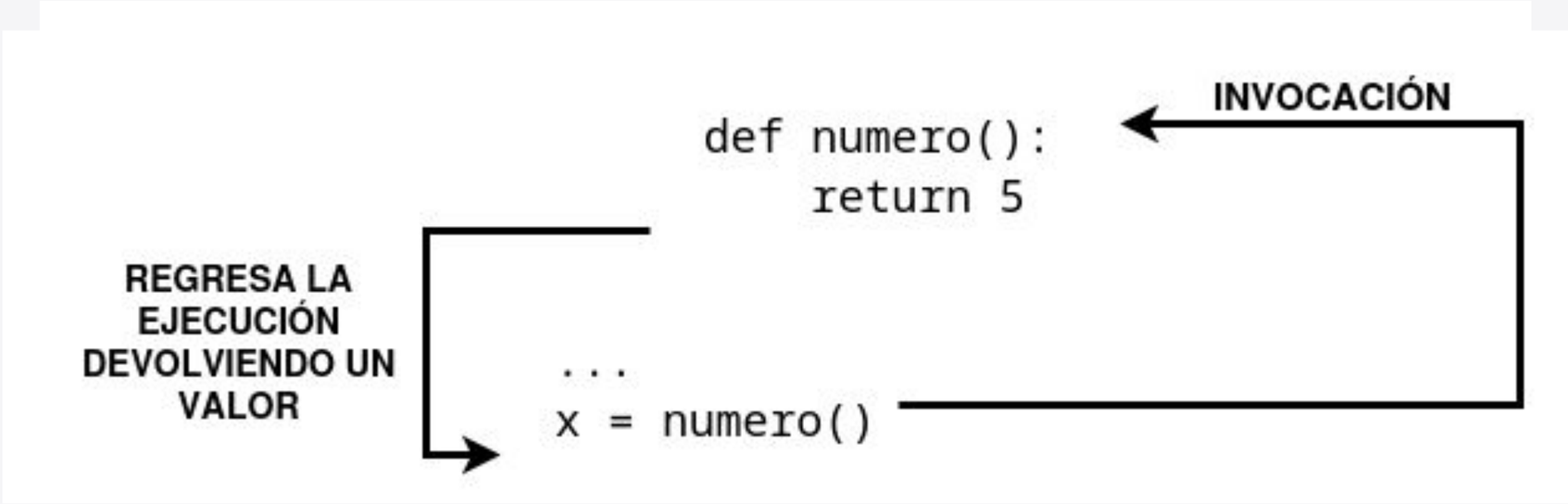
- **Ejercicio 3**
    - **Borrar**: Nos pide un nombre y si existe nos preguntará si queremos borrarlo de la agenda.
    - **Listar**: Nos muestra todos los contactos de la agenda.

Implementar el programa con un diccionario.

FUNCIONES


INVOCACIÓN
def mensaje():
print("Ingresa un valor: ")
...
mensaje()
a = int(input())
REGRESA LA EJECUCIÓN

FUNCIONES


INVOCACIÓN
def numero():
return 5
REGRESA LA EJECUCIÓN DEVOLVIENDO UN VALOR
...
x = numero()