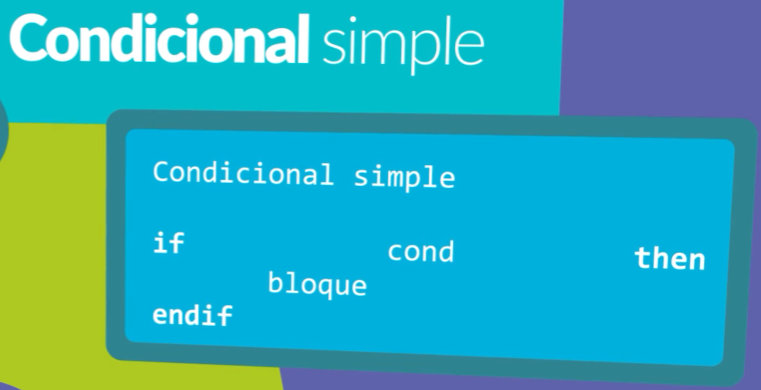
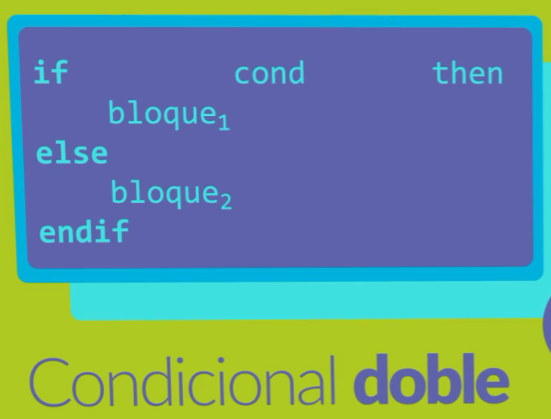
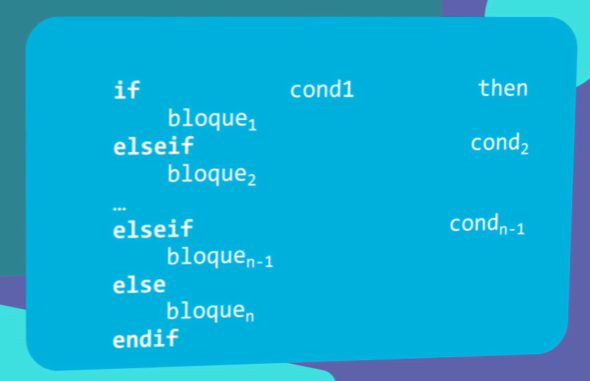
Módulo 3 – Estructuras de Control

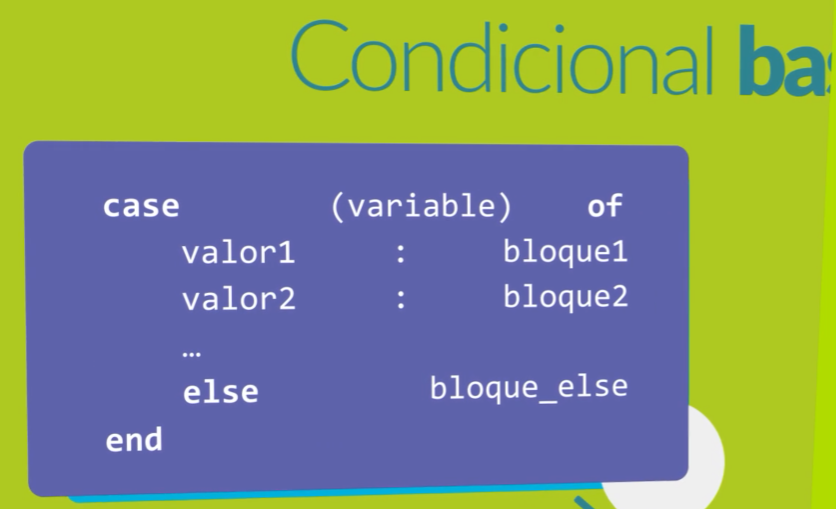
Lección 1: Estructuras de Control

1. Video 1: Estructuras de Control Condicional









1. Actividad 1

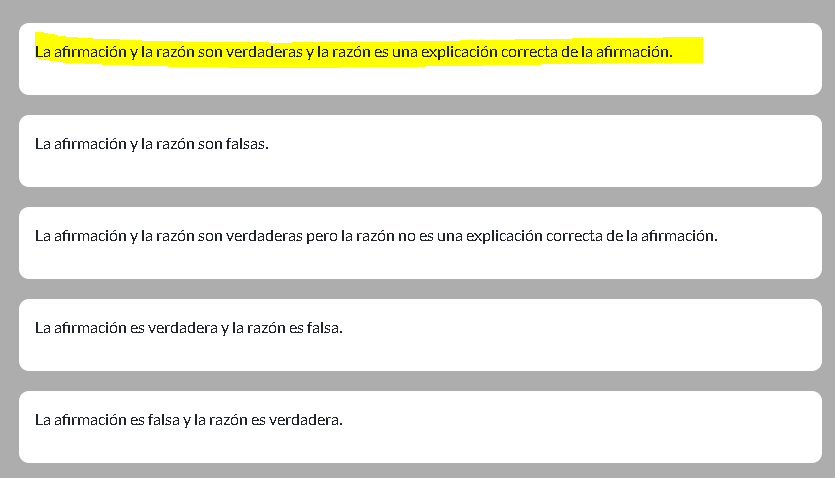
PRUEBA TUS  
**CONOCIMIENTOS**

¡Hola! Excelente momento para poner en práctica lo visto acerca de las estructuras de control condicionales. A continuación, te presentamos un texto que se divide en una afirmación y una razón que la sustenta. Te invitamos a leer el texto y escoger la opción correcta según lo estudiado hasta ahora.

**Afirmación**: Los condicionales son muy útiles en la programación para solucionar problemas

...porque...

**Razón**: muchas veces es necesario tomar acciones dependiendo de algunas condiciones, que se pueden dar durante la ejecución del programa.



1. Ejercicio 1: Condicionales

#

DESAFÍO  
PRÁCTICO

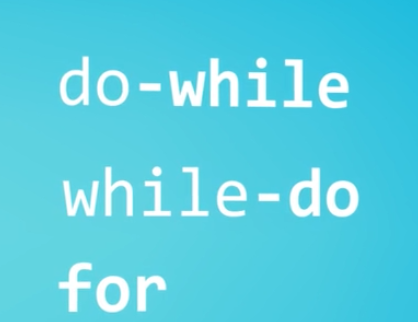
¡Saludos! Qué bien que vas a seguir poniendo en práctica lo visto en relación con las estructuras de control condicionales. En este ejercicio vamos a aplicar estos conocimientos adquiridos hasta el momento.

Escribe un programa en pseudocódigo que dadas tres criptomonedas y sus respectivas cantidades, imprima de forma ordenada decreciente cada moneda con sus respectivos valores.

Recomendaciones:

* Considera el uso de condicionales
* Recuerda las operaciones de lectura y escritura

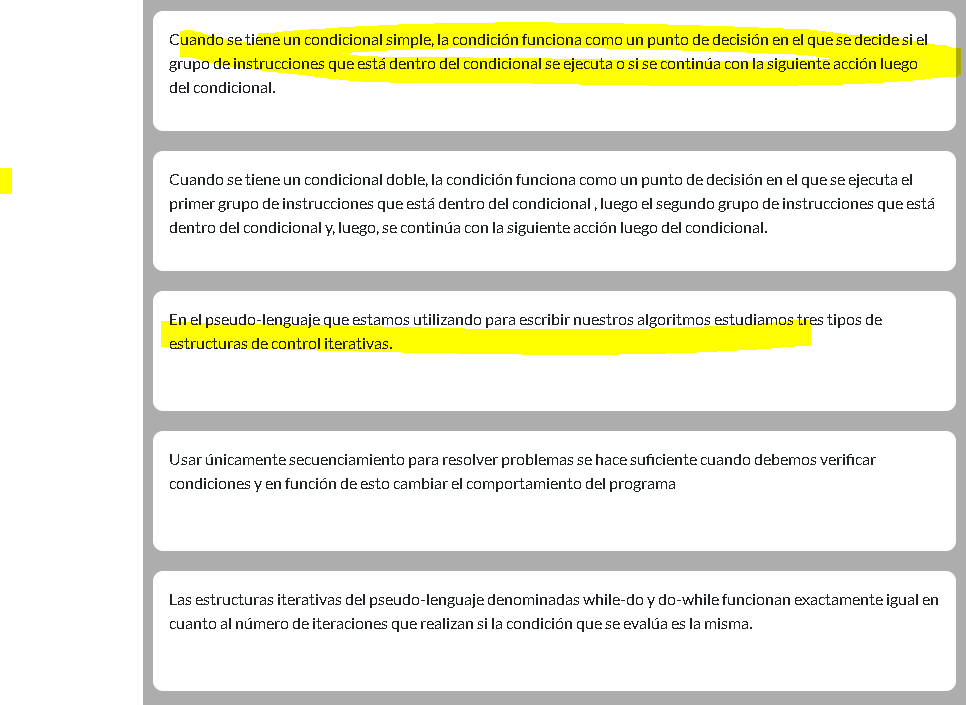
1. Video 2: Estructuras de Control iterativas



1. Actividad 2

PRUEBA TUS  
CONOCIMIENTOS

Ahora es el momento de consolidar los conocimientos adquiridos realizando esta actividad. Una vez estudiado las estructuras condicionales e iterativas del pseudo-lenguaje que estamos utilizando para expresar nuestros algoritmos; indica cuáles de las siguientes afirmaciones son las correctas:



1. Ejercicio 2: Ciclos

DESAFÍO  
PRÁCTICO

¡Hola! Excelente oportunidad para aplicar los conocimientos adquiridos hasta el momento, en este ejercicio vamos a consolidar lo visto en relación a los ciclos como estructuras de control; estos permiten realizar una serie de instrucciones de manera repetitiva con base en una condición, evitando el tener que copiar el mismo código muchas veces.

En tal sentido, te solicitamos que escribas un programa en pseudocódigo que reciba cinco criptomonedas, cada una con sus respectivas cantidades y precios en dólares americanos (USD), y luego imprima el valor total en USD que tienes acumulado.

Recomendaciones:

* Considera el uso de ciclos iterativos
* Recuerda las operaciones de lectura y escritura

1. Funciones

CÁPSULA DE  
CONOCIMIENTO

Funciones

¡Hola! Te felicitamos porque vas a muy buen ritmo. En este punto vamos a abordar los casos en que para la resolución de problemas, necesitamos ejecutar un grupo de instrucciones más de una vez pero en diferentes partes de nuestros algoritmos. Existen varias formas de solventar esta situación, dependiendo si sólo se necesita ejecutar las instrucciones, si necesitamos que a partir de esa ejecución se obtenga uno o más valores o si se requiere pasar uno o más valores a esas instrucciones para que puedan ser ejecutadas y obtener los resultados esperados.

En este sentido podemos definir y utilizar Funciones y Procedimientos. Estos mecanismos facilitan la estructuración y reutilización de instrucciones, evitando la duplicación y redundancia en nuestros algoritmos y promoviendo la reutilización de código. Una vez que se definen las funciones o procedimientos, éstos pueden ser invocados desde cualquier parte del algoritmo a través de su nombre, lo cual nos permite crear algoritmos más legibles y mejor estructurados. Ahora vamos a explicar con más detalle la definición y uso de funciones.

Una función es un grupo de sentencias que forman una expresión y recibe uno o más valores denominados argumentos, para producir un único valor que se denomina resultado. Este concepto se fundamenta en el concepto de función matemática; en el sentido, una función es una relación que asocia elementos de entrada pertenecientes a un dominio con uno y solo un elemento del codominio representado en su valor de retorno.

Las funciones se usan cuando existen dos o más porciones de algoritmo dentro de un código que son iguales o muy similares y, cuya ejecución, retorna un único valor. Esa porción se define como una función y se le asigna un nombre, para luego invocarla mediante ese nombre en cualquier parte del algoritmo y hasta dentro de la misma función u otras funciones. El tipo del valor de retorno de una función puede ser numérico, una cadena de caracteres y booleano.

En el seudo-lenguaje una función se declara de la siguiente manera:

funcion <nombre>(<listaParámetrosFormales>): <tipoResultado>  
   <declaraciones>  
inicio  
   <instrucciones>  
retorna <expresión>  
finfuncion

Donde:

<nombre> :nombre de la función por el cual será invocada

<listaParámetrosFormales>: contiene nombre y tipo de las variables que pasan alguna información necesaria para que la función ejecute el conjunto de instrucciones.

<tipoResultado>: indica el tipo de dato que devuelve la función

<declaraciones>: representa el conjunto de variables definidas para la función (diferentes a los parámetros)

<instrucciones>: representa el conjunto de instrucciones que realiza la función

<expresión>: representa el valor que retorna la función

Una vez que se ejecuta la función, se devuelve el control del programa al lugar donde se ha llamado a la función.

Dentro del contexto del proyecto que estamos desarrollando para tu portafolio, vamos a definir una función que compare las cotizaciones de dos criptomonedas e indique cuál es la más rentable, si tienen las misma rentabilidad que retorne la primera criptomoneda:

funcion MayorRentabilidad(crip1, crip2: string; cotiz1, cotiz2: float): string  
   var masRentable: string;  
   inicio  
&nsbp;&nsbp; if cotiz1 >= cotiz2  
     then masRentable = cotiz1;  
     else masRentable = cotiz2;  
   retorna masRentable;  
finFuncion

Para hacer uso de la función MayorRentabilidad se debe invocar por su nombre y asignarle el valor a una variable del tipo que retorna la función, por ejemplo:

var CriptoGanadoraHoy: string;  
   CriptoGanadoraHoy = MayorRentabilidad(bitcoin, ethereum, 0.0125, 0.0045)  
   escribe (CriptoGanadoraHoy);

En el ejemplo bitcoin, ethereum, 0.0125, 0.0045, se denominan parámetros actuales y son los valores con los cuales se ejecuta la función.

Hasta este punto hemos visto cómo y cuándo podemos definir y utilizar las funciones dentro de tus algoritmos, ahora te invitamos a seguir adelante ampliando tus conocimientos de programación.

1. Ejercicio 3: Funciones Pseudocódigo

DESAFÍO  
PRÁCTICO

¡Excelente verte de nuevo! En esta oportunidad vamos a aplicar los conocimientos vistos en relación a la definición de funciones en pseudo-código. Una de las ventajas de conocer estos conceptos es que son generales y, luego, se te hace mucho más directo hacer la traducción de tus soluciones en algún lenguaje de programación particular como por ejemplo Python.

Te presentamos la definición de una función escrita en pseudo-código denominada ConversionCriptomoneda (…) que permite convertir una cantidad de dinero dada en bitcoins y ripples a dólares americanos; retornando el monto total resultante de la suma de los dosvalores calculados en dólares americanos. Estas son las equivalencias aproximadas:

1 bitcoin = 7,442.50 USD$

1 ripple=0.660982 USD$

Puedes hacer uso del siguiente código base:

funcion ConversionCriptomoneda(cantBTC,cantXRP: float):  
   Var saldoTotalUSD,BTCUSD,XRPUSD: float;  
   BTCUSD=7442.50  
   XRPUSD=0.660982;  
  
   saldoTotalUSD;  
finFuncion

1. Ejercicio 4: Funciones

DESAFÍO  
PRÁCTICO

¡Saludos! En este ejercicio vamos a verificar la utilidad de incluir las funciones en nuestras soluciones algorítmicas. Para esto, te solicitamos escribir un programa en pseudocódigo que reciba cinco criptomonedas, cada una con sus respectivas cantidades y precios en dólares americanos (USD), y luego imprima el valor total en USD que tienes. La captura de los datos se debería poder utilizar en diferentes partes de un pseudocódigo más complejo.

Recomendaciones:

* Considera el uso de funciones
* Recuerda las operaciones de lectura y escritura

Lección 2: Estructuras de control condicional en Python

1. Estructuras de control condicional

CÁPSULA DE  
CONOCIMIENTO

Estructuras de control condicional

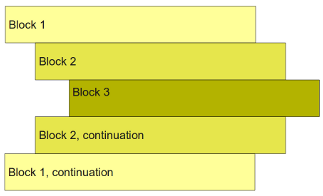
¡Hola! Continuamos profundizando en nuestro lenguaje de programación. En esta lectura comenzaremos a mostrarte las estructuras de control condicional if, las cuales permiten modificar el flujo de control de la ejecución de un programa Python en función de condiciones.

En general, una estructura de control o instrucción condicional selecciona un bloque de instrucciones a ejecutar con base en el cumplimiento de una condición. En este contexto es importante aclarar primero en qué consiste un bloque Python.

En la mayoría de los lenguajes de programación imperativos un bloque es un conjunto de instrucciones de un programa que se ejecutan como una unidad. Ejemplos de bloques son: módulos, cuerpo de funciones, definiciones de clases, entre otros. Generalmente, los bloques pueden contener otros bloques, obteniendo una estructura de programa anidada.

Los lenguajes de programación utilizan distintos mecanismos para definir bloques. Por ejemplo, lenguajes como Pascal y derivados utilizan las palabras claves begin y end para delimitar bloques; mientras que C, C++, C# y Java utilizan llaves ({ y }) para delimitar el inicio y fin de un bloque. En este tipo de lenguajes el sangrado o indentation del código se utiliza para legibilidad del mismo, pero no tiene significado sintáctico. En Python, a diferencia de la mayoría de los lenguajes imperativos, los bloques son conformados de manera sintáctica usando sangrado (o indentation, en inglés) basado en espacios en blanco o caracteres tab (tabulador), resultando un mecanismo obligatorio para agrupar código y que, en la mayoría de los casos, facilita la legibilidad y agrega simplicidad al no tener que usar elementos sintácticos que delimiten los bloques.

En Python, todas las instrucciones que están a misma distancia del comienzo de línea son parte del mismo bloque. Cada vez que se requiera anidar un nuevo bloque se anida más a la derecha con respecto del bloque anterior. De manera gráfica se observa en la siguiente figura:



En la medida que se vayan introduciendo las estructuras de control en Python se entenderá mejor la noción de bloque y la necesidad de anidamiento y sangrado de los mismos.

Condiciones

Las estructuras de control condicional introducidas en esta lectura utilizan condiciones lógicas, las cuales deben ser verificadas para determinar hacia dónde fluye el control del programa en Python. Las condiciones lógicas son expresiones cuyo resultado de evaluación es del tipo bool (True o False).

Las condiciones en Python pueden ser:

* Expresiones con operadores relacionales, por ejemplo: 2 < 5, x > 10, x == 10, a != b.
* Expresiones con operadores booleanos, por ejemplo: x >= 10 and x < 100, not x == y, a or b or c.
* Expresiones de funciones booleanas, por ejemplo: ‘3’ in "Python3", ‘$’ not in "Python3".

Como se indicó anteriormente, las condiciones lógicas son expresiones que se evalúan como True o False; sin embargo, en Python valores de otros tipos pueden evaluarse como booleanos. En particular, los siguientes valores u objetos son evaluados en Python como falso:

* Valores numéricos cero (0, 0.0, 0.0+0.0j)
* El valor booleano False
* Strings vacíos
* Listas, tuplas y diccionarios vacíos
* El valor especial None

Cualquier otro valor es evaluado como True.

**Condicionales simples:**

En Python un condicional simple tiene la siguiente sintaxis:

if <expresión condicional>:  
   <bloque de instrucciones>

donde <expresión condicional> es una condición lógica y <bloque de instrucciones> es un conjunto de instrucciones agrupadas como un mismo bloque de código. En general la semántica de esta estructura de control indica que el <bloque de instrucciones> será ejecutado siempre y cuando la <expresión condicional> se evalúe como True.

Un ejemplo del uso de este tipo de estructura de control es el siguiente:

if peso > 23:  
   print("Se aplicará un sobrecargo de 30 USD por el sobrepeso de equipaje.")  
   # sumar el cargo por sobre peso al total a pagar

En este ejemplo se aplica una tasa adicional a pagar como consecuencia de que un equipaje sobrepase los 23 kg. En caso contrario, no se aplica ningún sobrecargo.

**Condicionales dobles:**

En ocasiones, es necesario realizar una acción cuando se cumple una condición y una acción diferente si ésta no se cumple. En este caso, disponemos de una estructura de control condicional doble (o if-then-else, para simplificar):

if <expresión condicional>:  
   <bloque 1> else:  
   <bloque 2>

donde el <bloque 1> se ejecuta si la <expresión condicional> se evalúa como True y, en caso contrario, se ejecuta el <bloque 2>. Por ejemplo:

horasTrabajadas = int(input("¿Cuántas horas se trabajaron en la semana? "))  
if horasTrabajadas <= 40:  
   totalCobro = pagoHora\*horasTrabajadas  
else:  
   sobretiempo = horasTrabajadas - 40  
   totalCobro = pagoHora\*40 + 1.5\*pagoHora\*sobretiempo  
print("Total a cobrar: "+str(totalCobro))

En este ejemplo se hace un cálculo del total semanal a cobrar por un empleado. En este contexto, horasTrabajadas denota el total de horas trabajadas en la semana con un pago por hora de pagoHora y totalCobro es el total a calcular. Si el trabajador reporta más de 40 horas, las horas extras se pagan con 50% de incremento. Luego, la estructura condicional if-then-else permite considerar el caso normal (40 horas o menos) o el caso con sobre tiempo (más de 40 horas) y con base en eso se plantea la lógica para el cálculo.

**Expresión condicional:**

Muchos lenguajes ofrecen un mecanismo para escribir expresiones condicionales, es decir, expresiones que al ser evaluadas utilicen una lógica condicional para determinar el valor y que éste se pueda utilizar como lado derecho de una asignación o como parámetro a una función.

En Python se pueden escribir expresiones condicionales mediante la siguiente sintaxis:

<expresión 1> if <condición> else <expresión 2>

donde se evalúa <condición> y si es True se evalúa la <expresión 1> y en caso contrario, se evalúa la <expresión 2>. La evaluación es perezosa en el sentido de que la <expresión 2> sólo se evaluará si es necesario.

Por ejemplo, esta asignación:

max = a if (a > b) else b

es equivalente al bloque de código:

if a > b:  
   max = a  
else:  
   max = b

**Condicionales múltiples:**

Existen situaciones en las que requerimos más de dos casos para modelar una condición de ejecución condicional y, aunque las estructuras condicionales if-then-else se pueden anidar, es posible que resulte conveniente tener más de dos opciones en una única estructura condicional. Para esto se dispone en Python de la estructura condicional if-elif-else cuya sintaxis es la siguiente:

if <condición 1>:  
   <bloque 1>  
elif <condición 2>:  
   <bloque 2> …  
else:  
   <bloque n>

donde se irán comprobando en orden <condición 1>, <condición 2>, <condición n-1> y se ejecutará el bloque asociado a la primera que se evalúe en True. Si todas las condiciones fallan se ejecutará el <bloque n> asociado a la parte else.

Por ejemplo, supongamos que requerimos calcular el índice de masa corporal (IMC = masa kg / tamaño m2) para una persona y clasificarlo según lo indicado por la Organización Mundial de la Salud:

| Clasificación | IMC = peso / altura2 (kg/m2) |
| --- | --- |
| Bajo peso | menor a 18,5 |
| Normal | entre 18,5 y 25 |
| Sobrepeso | entre 25 y 30 |
| Obesidad Leve | entre 30 y 35 |
| Obesidad Media | entre 35 y 40 |
| Obesidad Mórbida | superior a 40 |

Para realizar este cálculo podemos escribir el siguiente programa en Python:

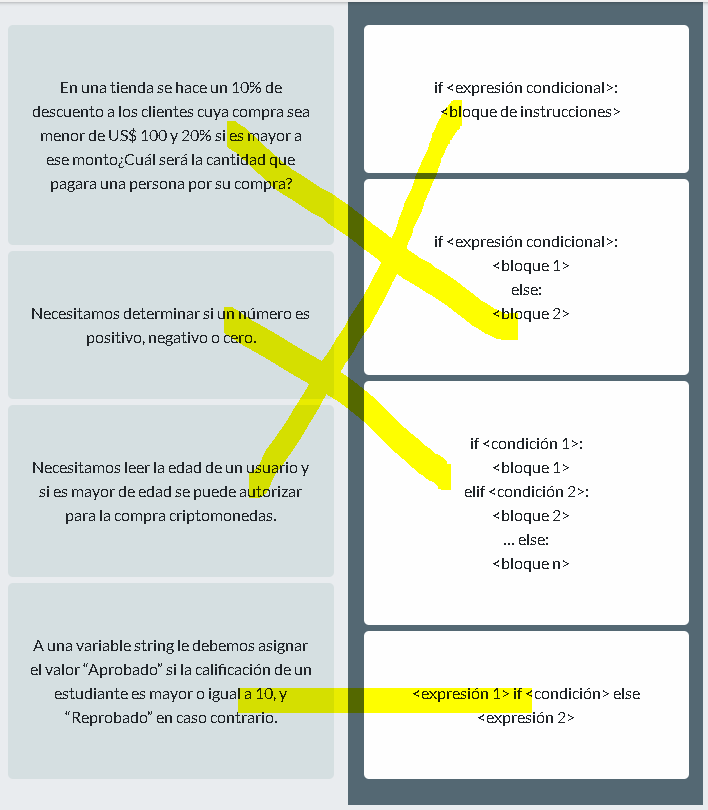
peso = float(input("Indique su peso (kg): "))  
altura = float(input("Indique su altura (mts): "))  
imc = peso / altura\*\*2  
if imc < 18.5:  
   print("Bajo peso")  
elif imc >= 18.5 and imc < 25:  
   print("Normal")  
elif imc >=25 and imc < 30:  
   print("Sobrepeso")  
elif imc >=30 and imc < 35:  
   print("Obesidad Leve")  
elif imc >= 35 and imc < 40:  
   print("Obesidad Media")  
else:  
   print("Obesidad Mórbida")

Con estas estructuras de control ya podemos plantear código Python donde algunos bloques se ejecuten según ciertas condiciones, expandiendo la posibilidad de problemas que podemos resolver. Sigue adelante ampliando tus conocimientos acerca de este maravilloso lenguaje de programación.

1. Actividad 1

ANALIZA Y  
RELACIONA

¡Felicitaciones! Estás avanzando a buen ritmo. A continuación, vamos a consolidar lo estudiado acerca de las estructuras condicionales en Python. Te ofrecemos dos columnas que contienen enunciados de problemas y estructuras de control condicionales; relaciona el problema de la columna de la izquierda con la estructura de control condicional que consideres apropiada incorporar en su solución que aparece en la columna de la derecha:



1. Ejemplo 1: Condiciones múltiples
2. Actividad 2

PRUEBA TUS  
CONOCIMIENTOS

¡Hola! Ahora es el momento perfecto para consolidar los conocimientos adquiridos con respecto a las estructuras condicionales que ofrece el lenguaje de programación Python. En esta actividad, te solicitamos que indiques cuáles de las siguientes sentencias son usadas en Python para construir condicionales:



1. Ejercicio 2: Condicionales

DESAFÍO  
PRÁCTICO

¡Hola! Estamos muy orgullosos que vayas a aplicar lo visto acerca de las estructuras de control condicionales que ofrece Python en este ejercicio. Nuevamente, podrás incrementar las soluciones de tu portafolio.

Escribe un programa en Python que dada una criptomoneda y un monto en esa criptomoneda, permita incrementar el saldo de tu teléfono móvil. Considera que los pagos pueden ser realizados mediante bitcoin (BTC), dash (DASH) o litecoin (LTC).

1. Actividad 3

PRUEBA TUS  
CONOCIMIENTOS

¡Saludos! Excelente momento para consolidar los conocimientos adquiridos con respecto a los condicionales provistos en Python realizando esta actividad. Ahora te solicitamos detectar todas las líneas que presentan errores del siguiente código:

1 if moneda=="BTC" or moneda=="DASH" or moneda=="LTC":

2 cantidad=float(input("Ingrese la cantidad a utilizar: "))

3 if moneda=="BTC"

4 monto=cantidad \* BTCUSD

5 elseif moneda=="DASH"

6 monto=cantidad \* DASHUSD

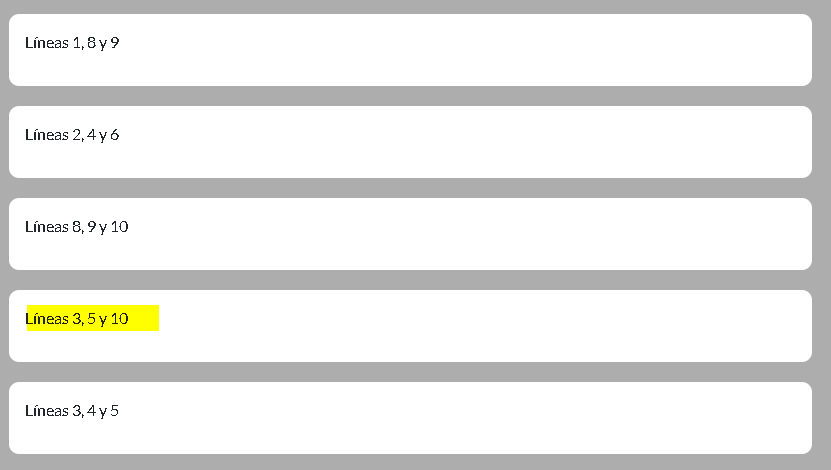
7 else:

8 monto=cantidad \* LTCUSD

9 print("La cantidad de USD recargado fue de: "+str(monto))

10 else

11 print("Error: Solo se permite utilizar BTC, DASH o LTC")



1. Ejercicio 3: Condicionales

DESAFÍO  
PRÁCTICO

¡Hola! Excelente oportunidad para poner en práctica lo visto acerca de las estructuras de control condicionales que ofrece el lenguaje de programación Python. Nuevamente, podrás incrementar las soluciones que puedes incorporar en tu portafolio.

En esta oportunidad, te solicitamos que escribas un programa en Python que lea el nombre de tres criptomonedas y el monto que posee el usuario de cada una de éstas; luego las muestre ordenadas de forma descendente cada nombre con su respectivo monto. ¡Éxito!

1. Actividad 4

PRUEBA TUS  
CONOCIMIENTOS

¡Hola! Has encontrado una excelente oportunidad para consolidar los conocimientos adquiridos con respecto a los condicionales presentes en el lenguaje de programación Python. Una vez estudiado estas estructuras condicionales; indica cuál es el funcionamiento del siguiente código en Python:

moneda1 = "BTC"

moneda2 = "LTC"

moneda3 = "BCC"

moneda = input("Ingrese la moneda a comprar: ")

if moneda == moneda1 or moneda==moneda2 or moneda==moneda3:

cantidad = float(input("Ingrese la cantidad de "+moneda+" a comprar:"))

if moneda == "BTC" and cantidad>0.01:

print("Compra Exitosa")

elif moneda == "LTC" and cantidad> 1.0:

print("Compra Exitosa")

elif moneda == "BCC" and cantidad > 0.1:

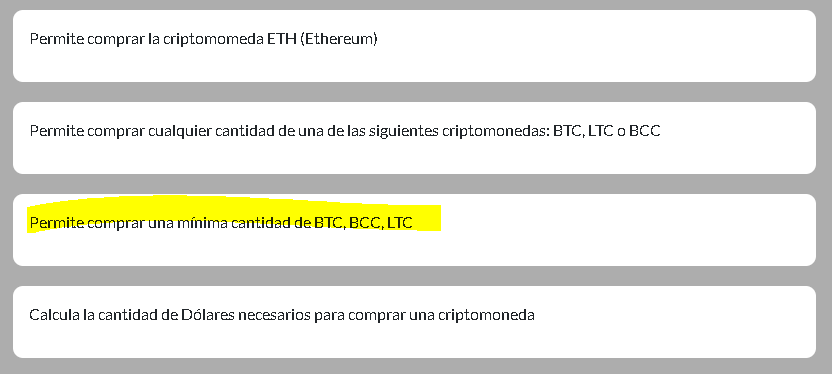
print("Compra Exitosa")

else:

print("Comprar fallida, cantidad menor al mínimo")

else:

print("Debe ingresar una de las siguientes monedas: BTC, LTC, BCC")



Lección 3: Estructuras de control iterativas en Python

1. Estructuras iterativas

CÁPSULA DE  
CONOCIMIENTO

Estructuras de Control Iterativas en Python

¡Qué bien que nos encontramos de nuevo! En esta oportunidad te mostraremos la estructuras de control iterativas del lenguaje de programación Python. En general, estas estructuras permiten realizar una secuencia de instrucciones de manera repetitiva (también conocidas como ciclos). En Python existen tres tipos de ciclos iterativos:

* Ciclos controlados por condiciones (while)
* Ciclos controlados por cantidad (for)
* Ciclos sobre colecciones

En esta lectura abordaremos los dos primeros tipos de ciclos. Los ciclos sobre colecciones e iteradores se muestran cuando estudiemos las colecciones de datos, en particular las listas.

**Instrucción while**

En Python una iteración while tiene la siguiente sintaxis:

while <expresión condicional>:  
<bloque>  
else:  
<bloque\_else>

donde <expresión condicional> es una condición lógica que determina la ejecución de las instrucciones que conforman <bloque>. Este bloque de instrucciones se repetirá mientras la condición se evalúe en True. Cuando la condición se evalúe en False no se ejecuta <bloque> y si existe la parte denotada por el else (que es opcional), se ejecuta <bloque\_else>. Esta última característica difiere este tipo de iteración con el disponible en otros lenguajes como C, C++ o Java.

Supongamos que dado un valor n queremos calcular el valor de su factorial (n!). Luego, utilizando un ciclo while tendríamos el siguiente programa:

n = int(input("Valor a calcular el factorial: "))  
elem = 1  
fact = 1  
while elem <= n:  
fact = fact \* elem  
elem = elem + 1  
print("El factorial de",n,"es",fact)

Nótese que la iteración se realizará variando el valor de elem desde 1 hasta n y acumulando en la variable fact el resultado de multiplicar todos los valores desde 1 hasta n. En este caso no es necesario un bloque para el no cumplimiento de la condición, por lo que prescindimos de la parte else del ciclo.

**Instrucción for**

Al igual que la instrucción while, el ciclo for es una instrucción iterativa de Python que permite repetir un bloque de código un número determinado de veces.

La sintaxis general de un ciclo for es la siguiente:

for <variable> in <secuencia>:  
<bloque>  
else:  
<bloque else>

donde la <secuencia> se evalúa una vez al inicio del ciclo y debe ser algún tipo de objeto iterable (un rango, una colección, etc). Cada elemento de la <secuencia> es almacenado en <variable> en cada iteración del ciclo, utilizando una operación de asignación y se ejecuta las instrucciones representadas por <bloque>. Cuando se haya realizado un ciclo por cada elemento de <secuencia> se ejecuta el <bloque else> (en caso de existir) y finaliza el ciclo for.

A manera de ejemplo, presentamos una versión del cálculo de factorial utilizando un ciclo for:

n = int(input("Valor a calcular el factorial: "))  
fact = 1  
for elem in range(1,n+1):  
fact = elem \* fact  
print("El factorial de",n,"es",fact)

En este caso se ejecuta el ciclo for n veces. La función range(a,b) retorna un iterador para el rango (a,b-1) con incremento 1. En caso de requerir un incremento diferente se puede utilizar un tercer parámetro quedando range(a,b,inc).

Existen otros tipos de secuencia que podemos iterar con un ciclo for. Esto será presentado cuando estudiemos las listas y secuencias en Python.

**Ciclos con sentencias break y continue**

Un ciclo en Python (while o for) puede contener también dos tipos particulares de instrucciones: break y continue. Una instrucción break finaliza el ciclo sin ejecutar la parte else mientras que la instrucción continue ignora el resto del ciclo y lo reanuda verificando de nuevo la condición.

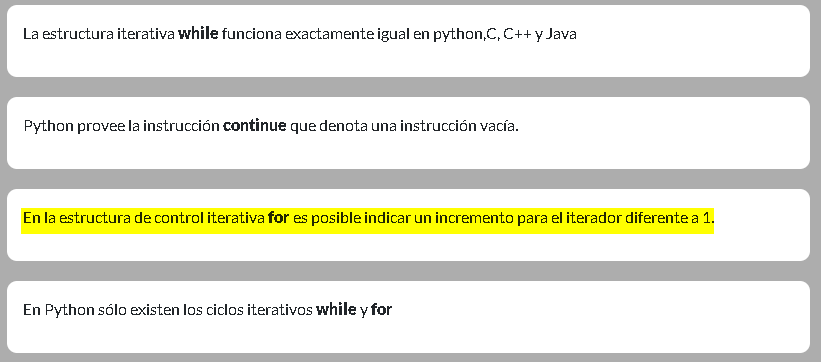
Adicionalmente, Python provee la instrucción pass que denota una instrucción vacía. Esta instrucción es muy útil para plantear bloques de código que no hacen nada, ya sea porque se requiere o porque es una porción de código sin programar aún y se desean hacer pruebas del resto del código.

Una vez conocidas estas estructuras de control iterativas hemos completado los conocimientos mínimos necesarios para escribir programas en Python. El contenido de esta lectura se puede complementar con lo que presentaremos acerca de las listas y colecciones que ofrece este lenguaje de programación, en las cuales el tratamiento iterativo es intrínseco y además añade algunas otras variantes, en particular a la iteración for. ¡Sigue adelante!

1. Actividad 1

PRUEBA TUS  
CONOCIMIENTOS

¡Hola! Felicitaciones por tu avance en el estudio del lenguaje de programación Python. En esta actividad vamos a poner en práctica los conocimientos adquiridos acerca de las estructuras iterativas de Python, indica cuál de las siguientes afirmaciones es la correcta:



1. Ejemplo 1: Iteraciones
2. Actividad 2

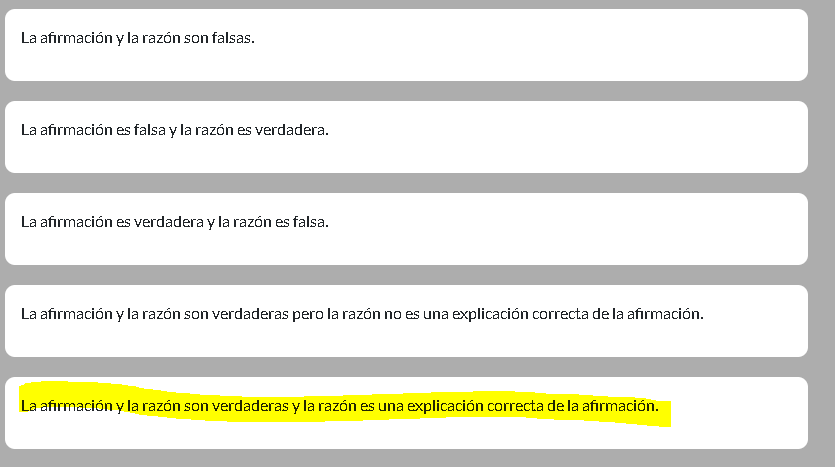
PRUEBA TUS  
CONOCIMIENTOS

¡Enhorabuena! Qué alegría que estés aquí para poner en práctica lo visto acerca de las estructuras de control iterativas que ofrece el lenguaje de programación Python. A continuación presentamos un texto que se divide en una afirmación y una razón que la sustenta. Lee el texto y escoge la opción correcta según lo estudiado hasta ahora.

Afirmación: Un ciclo en Python (while o for) puede contener también dos tipos particulares de instrucciones: break y continue.

...donde...

 Razón: una instrucción break finaliza el ciclo sin ejecutar el resto de las instrucciones del ciclo, mientras que la instrucción continue ignora el resto del ciclo y lo reanuda verificando de nuevo la condición.



1. Ejercicio 1: Iteraciones

DESAFÍO  
PRÁCTICO

¡Enhorabuena! Es admirable cómo has ido conformando tu portafolio, aquí tienes una nueva oportunidad de practicar lo visto y aumentar las soluciones que aplicarás en la solución de tu proyecto final.

Escribe un programa en Python que dados el nombre de una criptomoneda y su monto inicial para un usuario, genere un porcentaje aleatorio entre 0 y 3%, además puede ser de alza o baja, asociado a esa criptomoneda e imprima el porcentaje diario durante una semana, si es alza o baja, el saldo diario de la billetera digital del usuario y el total acumulado en esa semana.

1. Actividad 3

PRUEBA TUS  
CONOCIMIENTOS

¡Saludos! Felicitaciones porque has avanzado a un excelente ritmo. En esta actividad vamos a consolidar lo visto acerca de los ciclos iterativos que ofrece el lenguaje de programación Python. Analiza el siguiente código e identifica qué tarea ejecuta este programa en Python:

criptos = ["BTC","BCC","LTC","ETH","ETC"]

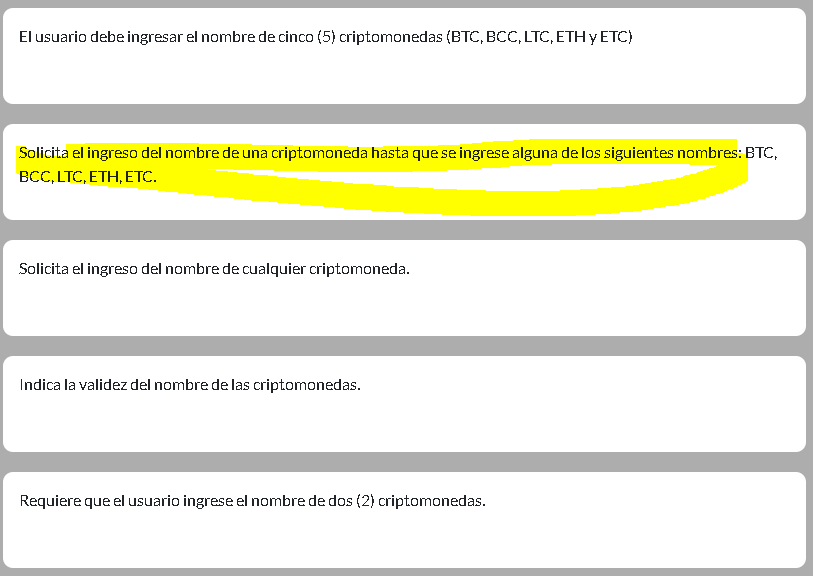
cripto = input("Ingrese el nombre de la moneda: ")

while cripto not in criptos:

cripto = input("La moneda "+cripto+" no existe. Ingrese el nombre de la moneda: ")

else:

print("Moneda Válida")



1. Ejercicio 2: Iteraciones

DESAFÍO  
PRÁCTICO

¡Enhorabuena! Te felicitamos por seguir enriqueciendo tu portafolio. Ahora tienes una nueva oportunidad para practicar lo visto y aumentar las soluciones que podrás incorporar en el desarrollo de tu proyecto final.

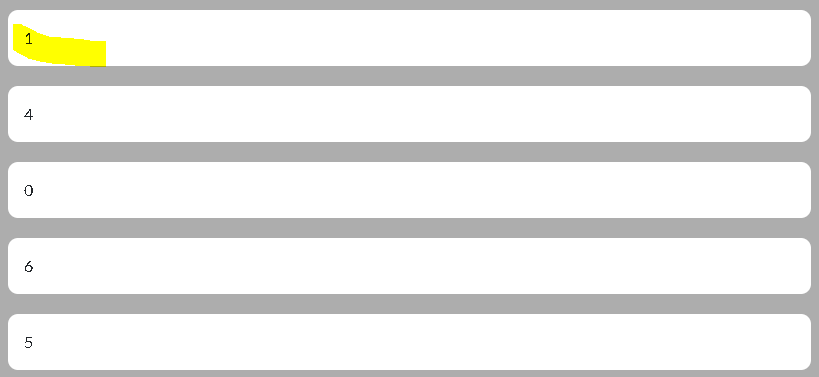
Escribe un programa en Python que solicite el ingreso de tres criptomonedas, las cantidades y cotizaciones en Dólares Americanos (USD) de cada una de éstas, validando que las tres criptomonedas pertenezcan a las siguientes (BTC, BCC, LTC, ETH, ETC y XRP), y que las cantidades y cotizaciones sean números reales. Por último el programa debe indicar la cantidad de USD que el usuario tiene acumulado.

1. Actividad 4

PRUEBA TUS  
CONOCIMIENTOS

¡Hola! Te damos la bienvenida a realizar esta actividad, donde podrás consolidar los conocimientos adquiridos con relación a las estructuras de control iterativas que ofrece el lenguaje de programación Python. En esta oportunidad te pedimo indicar qué valor se imprime una vez ejecutado el siguiente código en Python:

i=0  
while i<5:  
   i=i+1  
   break  
print(i)



Lección 4: Funciones en Python

1. Ejemplo 1: Funciones
2. Actividad 1

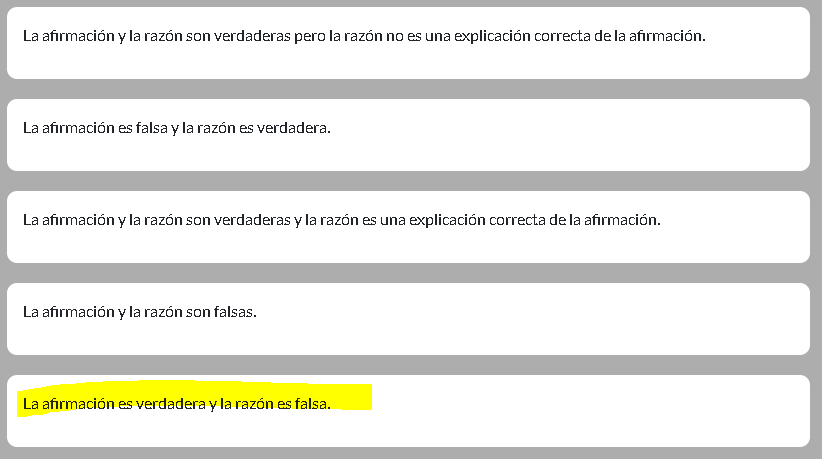
PRUEBA TUS  
CONOCIMIENTOS

¡Saludos! Qué gusto observar tu progreso y el esfuerzo que has invertido en realizar las actividades. A continuación te presentamos un texto que se divide en una afirmación y una razón que la sustenta. Te invitamos a leer el texto y escoger la opción correcta según lo estudiado hasta ahora con respecto a las funciones definidas por los programadores en Python.

Afirmación: Una función en Python que actúe como procedimiento tendrá un encabezado usando la palabra DEF, el nombre de la función y los parámetros formales

...pero...

Razón: retorna un valor vacío.



1. Funciones

CÁPSULA DE  
CONOCIMIENTO

Definición y uso de Funciones en Python

¡Hola! Seguimos conociendo estructuras fundamentales para la construcción de programas en Python. En esta lectura te mostraremos cómo se pueden crear nuevas funciones en este lenguaje de programación y cómo se usan. El uso aplica tanto para funciones definidas por el programador como para funciones predefinidas en el lenguaje o importadas de un módulo de código Python.

**Definición de funciones**

En los lenguajes de programación, las funciones son utilizadas para implementar porciones de código inspirado en las funciones matemáticas; es decir, calcular un resultado en función de sus parámetros. Al poseer un nombre, permite que se utilicen más de una vez en un mismo programa y que inclusive, sean reutilizadas en varios programas.

En Python, una función es definida mediante una instrucción def, cuya sintaxis general es la siguiente:

def <nombre>(<lista parámetros>):  
   <bloque>

donde <nombre> denota el nombre de la función definida. <lista parámetros> denota los parámetros o argumentos que recibe la función (también denominados parámetros formales) y <bloque> denota el cuerpo o instrucciones de la función, que es ejecutado cada vez que la función es llamada con los parámetros actuales correspondientes. Los parámetros pueden ser obligatorios u opcionales y de existir parámetros opcionales, éstos deben estar de últimos. El cuerpo de la función debe tener al menos una instrucción return, que denota la expresión resultante de la ejecución de la función y que será utilizada como resultado en el punto de la llamada a la función. Podemos tener funciones que no retornen valores (también denominadas procedimientos)

A manera de ejemplo, consideremos el cálculo del factorial de un número visto ahora como una función. En este caso se tiene la siguiente definición:

def factorial(n):  
   fact = 1  
   for elem in range(1,n+1):  
   fact = elem \* fact  
  return fact

Python permite la definición de funciones recursivas; es decir, funciones que se llaman a sí mismas. El cálculo de factorial es un buen ejemplo de una función que puede implementarse de manera recursiva, siendo más cercano a su definición matemática, pero (en este caso) de ejecución más ineficiente. La versión recursiva del cálculo de factorial en Python es la siguiente:

def factorial\_rec(n):  
   if n == 0:  
     return 1  
   else:

return n \* factorial\_rec(n-1)

Una vez definidas las funciones, éstas pueden ser llamadas en cualquier punto del programa utilizando el nombre de la misma seguida de los valores con los que se desea evaluar (parámetros actuales) entre paréntesis. Por ejemplo, para las dos definiciones presentadas para el cálculo de factorial se tienen los siguientes ejemplos de llamadas:

* n = int(input("Valor a calcular el factorial: "))
* print("El factorial de",n,"es",factorial(n))
* print("El factorial de",n,"es",factorial\_rec(n))

**Espacios de nombres (namespaces) y alcance**

Los namespaces son un sistema de nombramiento que evita la ambigüedad haciendo los nombres únicos. Un identificador o nombre es asociado al namespace en el que es definido y de esta forma el mismo identificador puede ser definido en distintos namespaces.

Estos son ejemplos de namespaces en Python:

* Identificadores globales de un módulo
* Identificadores locales en una función o método
* Identificadores de funciones predefinidas

Los namespaces tienen diferentes ciclos de vida, dependiendo del momento en que son creados (o destruidos). El namespace de las funciones predefinidas, por ejemplo, existe durante toda la ejecución del programa. El namespace de un módulo existe desde el momento que el módulo es importado hasta que el programa finaliza. Cada vez que una función es llamada, se crea un namespace local y es eliminado cuando la función finaliza.

El alcance (scope) de un identificador se refiere al área del programa donde el identificador puede ser utilizado sin ambigüedades. En Python ***el alcance se define de manera estática, es decir, está relacionado con la estructura del programa***, pero se usa de manera dinámica en la medida que la ejecución del programa fluye.

Durante la ejecución del programa los namespaces son enlazados formando una especie de pila y, al momento de utilizar un identificador, éste es buscando primero en los nombres locales del namespace en ejecución, luego en los namespaces de los bloques que engloban al bloque actual y así sucesivamente hasta alcanzar el namespace que contiene los nombres globales del módulo actual. El namespace que contiene los nombres predefinidos es el último en que se buscan los valores y siempre está disponible.

**Variables locales y globales**

En Python toda asignación a una variable dentro de una función es considerada una definición de una variable local. Por lo tanto, para utilizar un identificador global desde una función debe indicarse de manera explícita.

Supongamos el siguiente ejemplo:

def f():  
     x = 2  
   print(x)  
x = 3  
f()  
print(x)

Si tú eres un programador familiarizado con otros lenguajes, quizás pienses que la asignación x = 2 que está dentro de la función f afectará a la variable x global; sin embargo, Python al encontrar una asignación dentro de una función supone que corresponde a la definición de una nueva variable local; en consecuencia, la ejecución de este programa imprimiría los valores: 2 y 3.

Si deseamos asociar el uso de la variable global x dentro de la función f debemos indicar de manera explícita que ésta es global. A continuación un ejemplo:

def f():  
   global x  
   print(x)  
   x = 2  
x = 3  
f()  
print(x)

La sentencia global x dentro de la función f indica que todo uso o modificación de la variable x dentro de f afectará a la variable global, evitando que la asignación x = 2 cree una nueva variable local en f.

Con lo visto en esta lectura ya estamos listos para utilizar funciones definidas en módulos de Python (predefinidos o desarrollados por otros programadores), además ya podemos definir nuestras propias funciones e inclusive, nuestros propios módulos. Sigue adelante conociendo muchas más de las bondades de este lenguaje sencillo y poderoso.

1. Actividad 2

PRUEBA TUS  
CONOCIMIENTOS

¡Hola! Qué grato verte por aquí aplicando los tópicos estudiados en relación a la definción de funciones en Python. El resultado que obtengas al realizar esta actividad también puede ser incluido dentro de tu portafolio, el cual va creciendo en la medida que vas avanzando en este interesante recorrido de aprendizaje.

A continuación, se te ofrece una serie de sentencias, que debes ordenar para que definan una función correctamente en el lenguaje de programación Python.

1. ltcEUR=104.50

2. return totalEUR;

3. btcEUR=6500.81

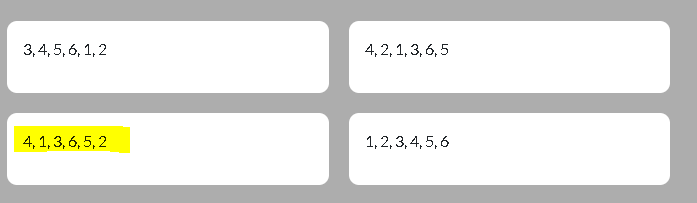
4. def ConversionCriptomoneda(bitcoin, ethereum, litecoin):

5. totalEUR=(bitcoin\*btcEUR)+(ethereum\*ethEUR)+(litecoin\*ltcEUR)

6. ethEUR=493.35

Escribe una función en Python denominada ConversionCriptomoneda (…) que permita convertir una cantidad de dinero dada en bitcoin, ethereum y litecoin a Euros; retornando el monto total resultante de la suma de los tres valores calculados en Euros. Estas son las equivalencias aproximadas:

* 1 bitcoin = 6,500.81 euros
* 1 ethereum =493.35 euros
* 1 litecoin = 104.50 euros



1. Ejercicio 1: Funciones

DESAFÍO  
PRÁCTICO

¡Saludos! Te felicitamos por lo bien que has creado insumos para enriquecer tu portafolio, aquí tienes una nueva oportunidad de practicar lo visto y aumentar las soluciones que te ayudarán en el desarrollo de tu proyecto final.

Ahora, te motivamos a escribir un programa en Python que solicite el ingreso de tres criptomonedas, las cantidades y cotizaciones en Dólares Americanos (USD) de cada una de éstas; además, debes validar que las tres criptomonedas pertenezcan a las siguientes (BTC, BCC, LTC, ETH, ETC y XRP), y que las cantidades y cotizaciones sean números reales. Por último, el programa debe indicar la cantidad de USD que el usuario tiene acumulada.

Te recomendamos hacer uso de funciones para realizar las validaciones.

1. Actividad 3

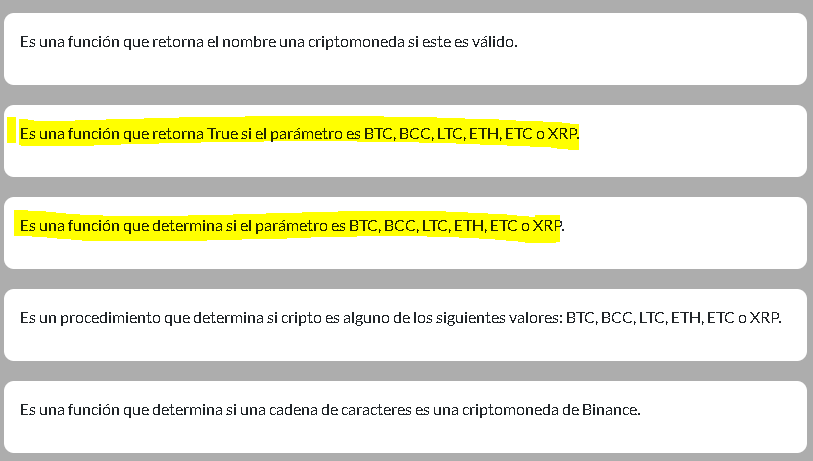
PRUEBA TUS  
CONOCIMIENTOS

¡Hola! Seleccionaste el momento perfecto para consolidar los conocimientos adquiridos realizando esta actividad. Vas a muy buen ritmo, ya estás entrando en aspectos más avanzados del lenguaje de programación Python. En tal sentido, luego de estudiar el uso de funciones en este lenguaje, analiza el siguiente código e indica cuál es el propósito de éste:

1. def esmoneda(cripto):

2. criptos = ["BTC","BCC","LTC","ETH","ETC","XRP"]

3. return cripto in criptos



1. Ejercicio 2: Funciones

DESAFÍO  
PRÁCTICO

¡Hola! Felicitaciones por seguir avanzando en la aplicación de los conceptos vistos. En este ejercicio pondrás en práctica lo referente a las funciones definidas por el programador en el lenguaje de programación Python; de esta forma sigues adquiriendo experiencia para plantear la solución de tu proyecto final.

Escribe un programa en Python que solicite el ingreso del nombre de una criptomoneda, validando que sea una de las siguientes (BTC, BCC, LTC, ETH, ETC y XRP), y que indique el precio actual de la misma obteniendo su valor de [Binance](https://www.binance.com/" \t "_blank) en USDT.

Para obtener el precio de la criptomoneda en USDT te recomendamos hacer uso de la API Rest de [Binance](https://www.binance.com/); además, puedes hacer uso de la API price ticker. Esta API se invoca en el navegador a través del URL: https://api.binance.com/api/v3/ticker/price?symbol=BTCUSDT.

Asimismo, utiliza el módulo resquests, para lo cual debes importarlo primero a la máquina usando pip3 install requests o pip install requests desde la consola de comando o terminal de tu sistema operativo.

Con la finalidad de invocar la API, puedes hacer uso de la función get de requests. Por ejemplo: requests.get(url).

Te recomendamos también hacer uso de funciones para realizar las validaciones.