Modulo 4 – Estructuras de Datos

Introducción

DESCRIPCIÓN

En general, el desarrollo de soluciones a problemas de la vida real requiere modelar datos con cierto grado de complejidad, para lo cual no basta con los tipos de dato simples que sólo permiten almacenar un valor y que existen en el pseudolenguaje y en los lenguajes de programación como Python. En estos casos, requerimos de tipos de dato estructurados o estructuras de datos puedan recolectar varios valores simultáneamente. Estos valores pueden ser, a su vez, del mismo tipo o de tipos diferentes; puede tratarse de estructuras cuya longitud puede variar durante la ejecución de los programas o estructuras de longitud fija. Todo ese mundo de las estructuras de datos lo vamos a conocer en esta oportunidad, para contar con muchas más herramientas que nos permitan plantear soluciones cada vez más eficientes. Anímate a se descubrir estos nuevos conceptos y a aplicarlos en tus programas para aumentar tu portafolio de soluciones. ¡Acompáñanos!

OBJETIVOS

* Identificar los tipos de dato estructurados presentes en el pseudolenguaje
* Aplicar la interacción de los programas con archivos secuenciales para leer y escribir datos
* Desarrollar programas que utilicen los diferentes objetos de datos estructurados que ofrece Python: listas, tuplas, conjuntos y diccionarios
* Usar operaciones y funciones asociadas a los datos estructurados en Python

Lección 1: Tipos de datos estructurados

1. Video: Tipo de datos estructurados

Estructuras de datos:

* Registros

Secuencia finita de elementos de tipos generalmente diferentes, asociados a los campos.

Record nombre

…

…

endRecord

* Arreglos

Secuencia finita de elementos del mismo tipo 🡪 Índice, array

Arreglos multidimensionales. Ej: float array matriz [n,n]

* Archivos secuenciales

Secuencia de elementos del mismo tipo, archivos secuenciales almacenados en disco

File nombre\_variable

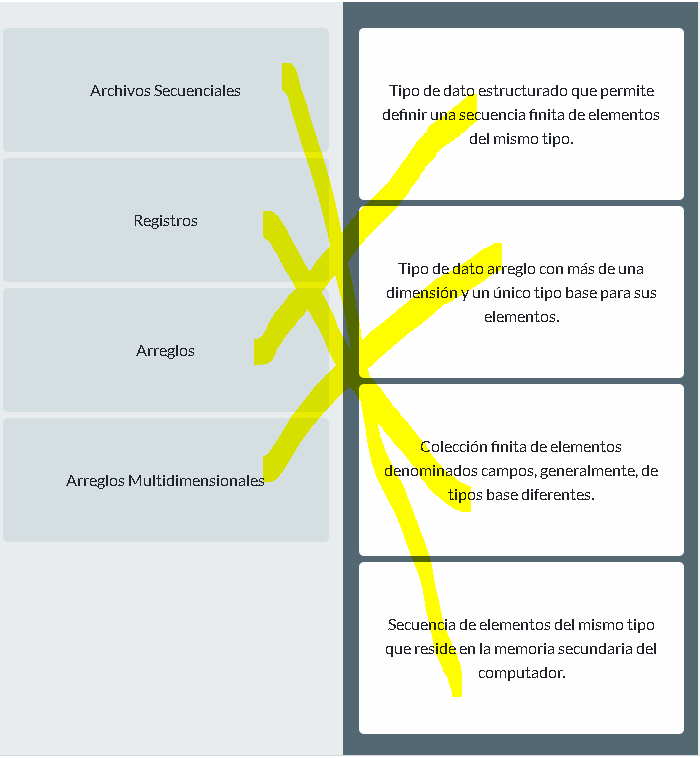
Open para leer o escribir un archivo

Close

1. Actividad 1

ANALIZA Y  
RELACIONA

¡Hola! Excelente que estés aquí para consolidar los conceptos relacionados con las estructuras de datos que puedes utilizar en tus algoritmos. A continuación se te ofrecen dos columnas que contienen conceptos acerca de datos estructurados en el pseudolenguaje, relaciona el tipo de dato de la columna de la izquierda con su correspondiente definición que aparece en la columna de la derecha:



1. Ejercicio 1: Estructuras de datos

DESAFÍO  
PRÁCTICO

¡Felicitaciones! Has tenido un gran avance. En este ejercicio pondrás en práctica lo visto acerca de las estructuras de datos denominadas arreglos, las cuales están presentes en la mayoría de los lenguajes de programación con el mismo nombre o se pueden simular a través de otras estructuras.

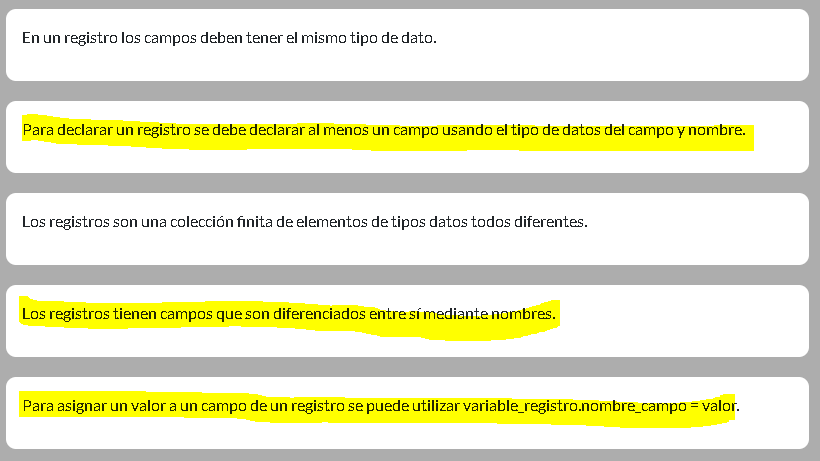
En esta oportunidad te pedimos escribir un programa en pseudocódigo que registre el nombre de cinco criptomonedas, con sus respectivas cantidades y precios en USD, haciendo uso de arreglos. Posteriormente, imprima los datos de cada moneda luego de ser ingresadas por el usuario.

Recuerda que debes hacer uso de las sentencias de lectura y escritura para que el usuario indique los datos a registrar.

1. Actividad 2

PRUEBA TUS  
CONOCIMIENTOS

¡Hola! Excelente momento para poner en práctica lo visto en relación a los tipos de datos estructurados en pseudocódigo, a saber: arreglos, arreglos multidimensionales, registros y archivos secuenciales. Considerando los registros, indica cuáles de las siguientes afirmaciones son las correctas:



1. Ejercicio 2: Estructuras de datos

DESAFÍO  
PRÁCTICO

¡Hola! Te felicitamos por el ritmo que estás llevando en tu formación. En esta oportunidad vamos a ejercitar lo visto acerca de una de las estructuras de datos que puedes definir en nuestro pseudocódigo.

Escribe un programa en pseudocódigo que registre el nombre de cinco criptomonedas, con sus respectivas cantidades y precios en USD, haciendo uso de registros. Luego, imprima los datos de cada moneda una vez que han sido ingresadas por el usuario. Además, debe indicar el monto total en USD que posee el usuario.

Recuerda que debes hacer uso de las sentencias de lectura y escritura para que el usuario indique los datos a registrar.

Lección 2: Listas en Python

1. Video: Listas

Existe una noción de orden, acceso a los elementos es mediante índice. Sus elementos pueden ser modificados. Función len() para obtener su longitud.

Los ciclos for están hechos para funcionar con listas.

1. Actividad

ANALIZA Y  
RELACIONA

¡Enhorabuena! A continuación, te presentamos la primera fila de una tabla que contiene algunas de las operaciones y su descripción que puedes aplicar al tipo de dato lista que ofrece el lenguaje de programación Python; necesitamos de tu ayuda para completar la tabla colocando cada celda en el lugar apropiado:



1. Ejercicio: Listas

DESAFÍO  
PRÁCTICO

¡Felicitaciones! Excelente momento para poner en práctica lo visto acerca de las listas que ofrece el lenguaje de programación Python y así podrás seguir enriqueciendo tu portafolio de soluciones.

Escribe un programa en Python que registre el nombre de cinco criptomonedas, con sus respectivas cantidades y precios en USD, haciendo uso de listas. Luego, imprima los datos de cada moneda una vez que han sido ingresadas por el usuario.

Recuerda que debes hacer uso de las sentencias de lectura y escritura para que el usuario indique los datos a registrar.

Listas

Listas CÁPSULA DE  
CONOCIMIENTO

**Listas en Python (tipo list)**

¡Hola! Te damos la bienvenida a conocer un poco más de uno de los tipos estructurados más utilizados en los programas escritos en Python. Los elementos de una lista conforman una secuencia indexada donde el primer elemento de la lista ocupa la posición 0 de la secuencia.

**Creación y manipulación de listas con sub-listas**

En Python podemos manipular también sub-listas (slices):

lista = ['a', 'b', 'c', 'd']

print(lista[1:3]) # imprime ['b', 'c']

print(lista[2:]) # imprime [‘c’,’d’]

print(lista[:2]) # imprime [‘a’,’b’]

print(lista[::-1]) # imprime [‘d’,’c’,’b’,’a’]

lista[0:2] = 'z' ## reemplaza ['a', 'b'] con ['z']

print(lista) ## imprime ['z', 'c', 'd']

Los slices en listas Python constituyen una característica muy poderosa en la solución de problemas que requieran el uso de listas.

**Otra forma de manipulación de listas en ciclos for**

Otra manera de iterar sobre los elementos de una lista es mediante un rango:

for i in range(0,len(lista)):

print(lista[i])

1. Adicionalmente, podemos utilizar el operador in para verificar si un elemento pertenece (o no) a una lista:

if ‘b’ in lista:

print(“Encontrado”)

if ‘z’ not in lista:

print(“No encontrado”)

**Funciones y operadores adicionales sobre listas**

A continuación, listamos algunas operaciones adicionales sobre listas:

| **Operación** | **Descripción** |
| --- | --- |
| **s.append(x)** | Agrega el valor x al final de la lista s |
| **s.pop(i)** | Retorna el elemento que se encuentra en la posición i de la lista s, eliminándolo de s |
| **+** | Sobrecarga el operador + para concatenar listas |
| **+=** | Sobrecarga el operador de incremento += para concatenar listas a una lista dada |
| **s.remove(x)** | Elimina la primera aparición de x en la lista s |
| **s.index(x[, i[, j]])** | Determina la posición de x en la lista s. Si se especifica el valor i, éste indica la posición a partir de la cual se buscará y si se indica el valor j, indicará la posición final de la búsqueda. En caso de no encontrar el valor x, se dispara una excepción |
| **s.insert(i,o)** | Inserta el objeto o en la posición i de la lista. Si la posición i no existe en la lista s, el objeto o es insertado de último |
| **s.sort()** | Ordena la lista s |
| **s.reverse()** | Invierte la lista s |

Excelente que ya cuentas con las herramientas necesarias para incorporar las listas en tus programas Python. Las listas son parte fundamental de Python y son utilizadas para resolver muchos problemas. Sigue adelante descubriendo otros conceptos muy útiles de este poderoso lenguaje de programación.

1. Ejemplo: Listas

Lección 3: Tuplas y conjuntos en Python

1. Video: Tuplas y conjuntos

Tupla: tipo tuple 🡪 Son secuencias de elementos de tamaño fijo encerrado en (), similar a las listas, es inmutable

Conjuntos: tipo set 🡪 Un conjunto es colección no ordenada de valores únicos. Operaciones: unión, intersección y diferencia de conjuntos

Conjunto = set() 🡪conjunto vacío

Ej:

conj\_a = {1}

conj\_b = {1,2,3}

conj\_c = set([‘a’,’b’,42])

Operaciones de conjuntos. Variable tipo set

s.add(e), agrega el elemento al conjunto s, si el elemento ya existe en el conjunto s no pasa nada.

s.clear(), elimina todos los elementos del conjunto s

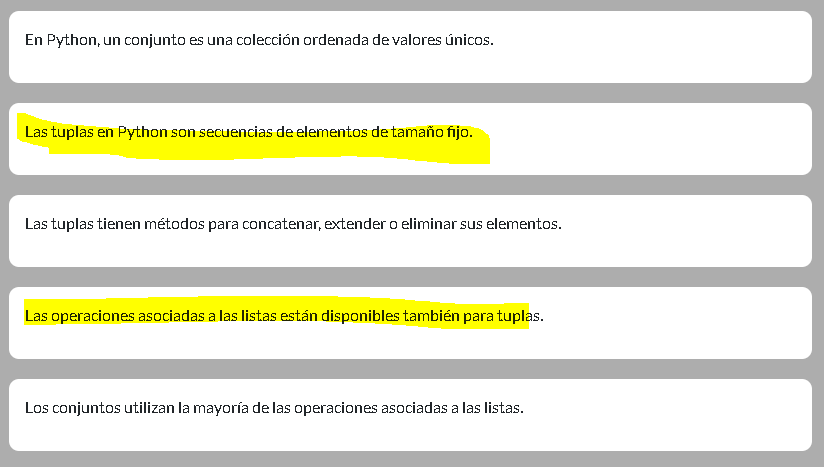
s.copy(), returna una copia del conjunto s

s.remove(e)

1. Actividad

PRUEBA TUS  
CONOCIMIENTOS

¡Hola! Felicitaciones por tus avances en el estudio de los tipos de datos estructurados de Python; específicamente en lo referente a los tipos denominados tuplas y conjuntos, sus características y operaciones. Te invitamos a que, considerando los conceptos visto, indiques cuáles de las siguientes afirmaciones son las correctas:



1. Ejercicio: Tuplas y conjuntos

¡Hola! Te felicitamos por llegar hasta este punto, aquí tienes una nueva oportunidad para incorporar nuevos elementos a tu portafolio de soluciones y adquirir más experiencia en el desarrollo de programas en Python.

En esta ocasión te solicitamos que escribas un programa en Python que verifique si el nombre de una criptomoneda, ingresada por el usuario, existe en coinmarketcap.com. Para ello recuerda utilizar el módulo requests, usando la API <https://api.coinmarketcap.com/v2/listings/> . Esta API retorna un json con una lista de datos de las criptomonedas, entre estos datos se encuentra el symbol de la criptomoneda que corresponde al nombre código de la moneda.

Te recomendamos hacer uso de tuplas para hacer la búsqueda del nombre de la criptomoneda ingresada por el usuario; esto debido a que hacer búsquedas en tuplas es mucho más rápido que en listas y así obtendrás una solución eficiente.

Consideraciones especiales:

1. Para hacer uso de las APIs de coinmarketcap.com se debe usar un API key, que se obtiene al registrase en: <https://coinmarketcap.com/api/> usando el plan Basic que es gratuito.
2. Luego de registrase ingresar a <https://pro.coinmarketcap.com/account,> colocar el ratón  sobre la sección API Key (Asteriscos) y dar click en el botón COPY KEY.
3. En el código Python usar una variable headers, para pasar los parámetros de autenticación con el API Key. Por ejemplo:

headers = { 'Accepts': 'application/json', 'X-CMC\_PRO\_API\_KEY': 'COLOCAR API KEY COPIADA'}

1. En la invocación del método get además del URL se deben pasar el headers y los parámetros que sean necesarios. Por ejemplo:

parametros = {'symbol': symbol}

requests.get("https://pro-api.coinmarketcap.com/v1/cryptocurrency/quotes/latest",headers=headers,params=parametros)

1. Tuplas y conjuntos en Python

Tuplas (tipo tuple) y conjuntos (tipo set) en Python

¡Qué gran gusto saludarte de nuevo! En esta oportunidad conoceremos más tipos de datos estructurados que provee Python para manejar secuencias con características particulares: las tuplas (secuencias de tamaño fijo) y conjuntos (colección no ordenada de elementos no repetidos).

Tuplas

Las tuplas -a diferencia de las listas- son consideradas estructuras inmutables; sin embargo, su manipulación es muy similar a la de las listas en Python.

A continuación, se enumera algunas diferencias de las tuplas con respecto a las listas:

* Las tuplas no tienen métodos para concatenar, extender o eliminar sus elementos (son de tamaño fijo). Sin embargo, sí es posible concatenar dos o más tuplas para crear una nueva.
* En general, las tuplas son más rápidas que las listas.
* Sirven para proteger datos que no queremos que sean modificados durante la ejecución del programa.

Funciones y operadores adicionales sobre tuplas

A continuación, listamos algunas operaciones específicas para tuplas; notemos que las operaciones definidas para listas y que no sean incompatibles con lo descrito en los párrafos anteriores, están disponibles para tuplas:

| **Operación** | **Descripción** |
| --- | --- |
| **cmp(t1,t2)** | Compara las tuplas t1 y t2 |
| **tuple(l)** | Convierte la lista l en una tupla |
| **max(t)** | Retorna el máximo elemento de la tupla t |
| **min(t)** | Retorna el mínimo elemento de la tupla t |

Conjuntos

Para crear conjuntos no vacios, basta con indicar los elementos que los conforman mediante una serie de elementos separados por comas (,) y encerrados entre llaves ({ y }):

conj\_a = { 1 }

conj\_b = { 1, 2, 3 }

conj\_c = set(['a', 'b', 42]) # crea un conjunto a partir de una lista

Funciones adicionales sobre conjuntos

Al ser un tipo de dato para manejar secuencias, los conjuntos utilizan la mayoría de las operaciones descritas para listas. A continuación se listan algunas funciones específicas para conjuntos:

| **Operación** | **Descripción** |
| --- | --- |
| **s.difference(t)** | Retorna la diferencia entre los conjuntos s y t. El operador – implementa esta función también por lo tanto, la expresión s – t es equivalente |
| **s.difference\_update(t)** | s = s – t |
| **s.discard(e)** | Elimina e del conjunto s. Si e no pertenece al conjunto, no ocurre ningún cambio |
| **s.union(t)** | Retorna un nuevo conjunto que contiene la unión de s y t. Se puede representar de manera abreviada como s | t |
| **s.intersection(t)** | Retorna un nuevo conjunto que contiene la intersección de s y t. Se puede abreviar como s & t |
| **s.isdisjoint(t)** | Retorna True si la intersección entre s y t es vacía |
| **s.pop()** | Retorna y elimina un elemento arbitrario del conjunto s. Si el conjunto s es vacío ocurre un error |
| **s.issubset(t)** | Verifica si todos los elementos de s pertenecen a t. Se puede abrebiar como s <= t |
| **s.issuperset(t)** | Verifica si todos los elementos de t pertenecen a s. Se puede abrebiar como s >= t |
| **s < t** | s <= t and s != t |
| **s > t** | s >= t and s != t |
| **s.symmetric\_difference(t)** | Retorna los elementos que pertenecen a s o a t pero no a ambos. Se puede abreviar como s ^ t |
| **s.remove(e)** | Elimina el elemento e de s. Ocurre un error si e no pertenece a s. |

Con todo este contenido hemos ampliado los conocimientos necesarios para el manejo de datos estructurados en Python, ofreciendo un mundo de posibilidades para la construcción de soluciones a problemas de mediana a alta complejidad. Sigue avanzando, hasta pronto.

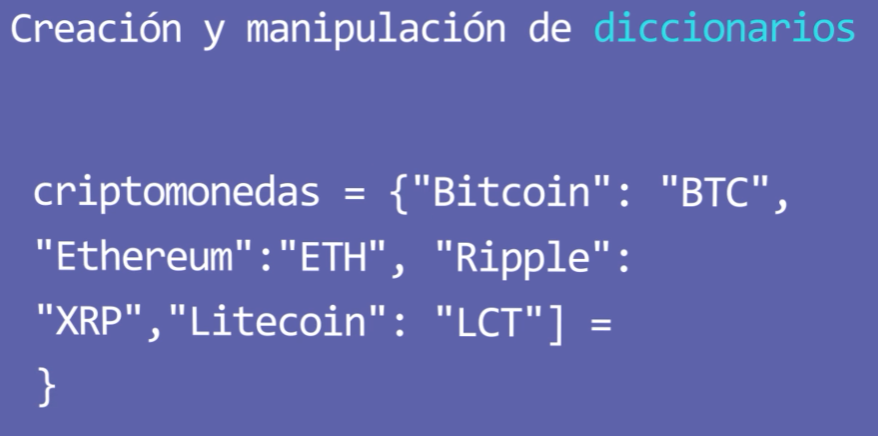
1. Ejemplo: Tuplas y conjuntos

Lección 4: Diccionarios en Python

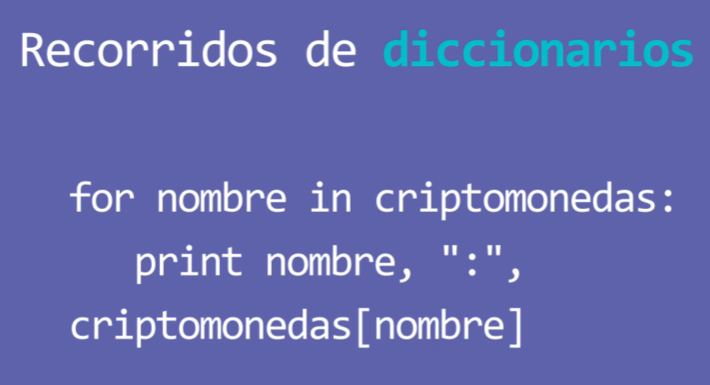
1. Video: Diccionarios

Tipo dict() – Permite guardar un conjunto no ordenado de pares claves – valor (key-value)

Permite una forma mucho más rápido de acceder, tuplas son inmutables, no se pueden modificar las llaves pero si los valores



Se utiliza la clave elegida en lugar del índice para acceder al valor de una clave.



diccionario = dict()

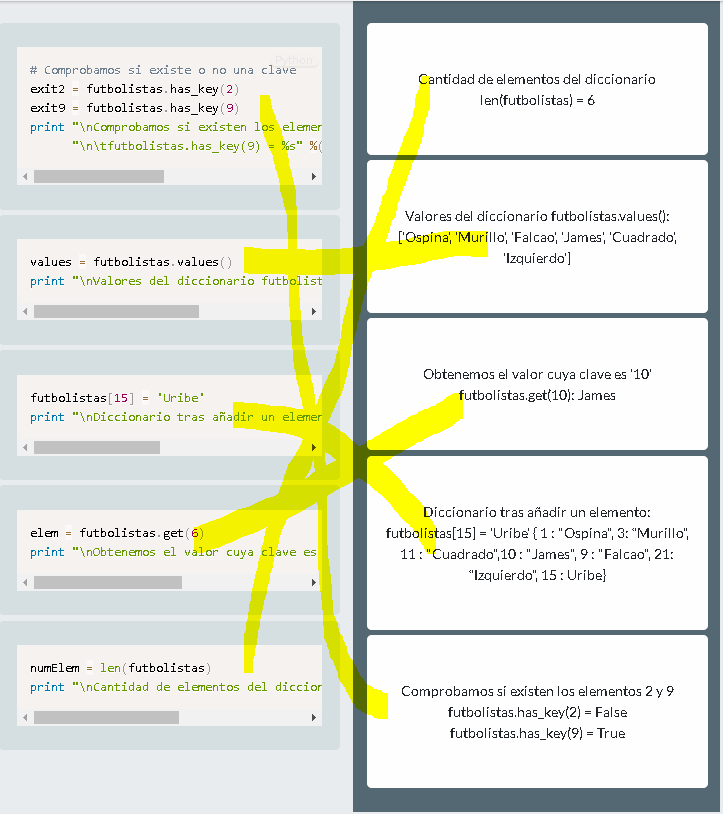
len(dict)

1. Actividad

**A**NALIZA Y  
RELACIONA

¡Hola! Esperamos que hayas disfrutado lo visto acerca de los diccionarios de Python. A continuación vamos a poner en práctica lo visto acerca de este concepto. Te ofrecemos dos columnas, la primera contiene unas sentencias donde se manipula de una variable de tipo diccionario denominada futbolistas y la columna de la derecha describe los resultados de la ejecución de esas sentencias. Te solicitamos que relaciones cada sentencia con su correspondiente resultado. Considera los siguientes valores para la variable futbolistas:

**futbolistas= { 1 : "Ospina", 3: “Murillo”, 11 : "Cuadrado",10 : "James", 9 : "Falcao", 21: “Izquierdo”}**



1. Ejercicio: Diccionarios

¡Saludos! Felicitaciones por seguir profundizando en los conceptos fundamentales del lenguaje de programación Python. Esta es una oportunidad única para aplicar lo aprendido y proponer soluciones que podrás incorporar a tu portafolio.

Escribe un programa en Python que solicite al usuario el nombre de una criptomoneda y verifique si ésta existe en coinmarketcap.com. Posteriormente, el programa debe imprimir el nombre abreviado o, como también se conoce, nombre código y el nombre completo de la criptomoneda. Para esto, recuerda utilizar el módulo requests, usando la API <https://api.coinmarketcap.com/v2/listings/> que retorna un json con una lista de datos de las criptomonedas. Entre estos datos tenemos el symbol de la criptomoneda que corresponde al nombre código y name que contiene el nombre completo de la criptomoneda.

Te recomendamos hacer uso de variables tipo diccionario y tuplas como claves.

Consideraciones especiales:

* Para hacer uso de las APIs de coinmarketcap.com se debe usar un API key, que se obtiene al registrase en: <https://coinmarketcap.com/api/> usando el plan Basic que es gratuito.
* Luego de registrase ingresar a <https://pro.coinmarketcap.com/account,> colocar el ratón  sobre la sección API Key (Asteriscos) y dar click en el botón COPY KEY.
* En el código Python usar una variable headers, para pasar los parametros de autenticación con el API Key. Por ejemplo:

headers = { 'Accepts': 'application/json', 'X-CMC\_PRO\_API\_KEY': 'COLOCAR API KEY COPIADA'}

JavaScript

* En la invocación del método get además del URL se deben pasar el headers y los parametros que sean necesarios. Por ejemplo:

parametros = {'symbol': symbol}

requests.get("https://pro-api.coinmarketcap.com/v1/cryptocurrency/quotes/latest",headers=headers,params=parametros)

1. Diccionarios en Python

CÁPSULA DE  
CONOCIMIENTO

Diccionarios en Python (tipo dict)

¡Excelente encontrarnos de nuevo! Seguimos profundizando en los tipos de dato estructurados que te ofrece Python. Ahora vamos a conocer la estructura de datos conocida como diccionario.

Los diccionarios son un tipo de estructuras de datos que permite guardar un conjunto no ordenado de pares clave-valor o, como se conoce en inglés key-value.

Las claves son únicas dentro de un mismo diccionario (es decir que no pueden existir dos elementos con una misma clave), pero el valor si se puede repetir. Los valores en un diccionario pueden ser de cualquier tipo.

Las tuplas pueden usarse como claves si solamente contienen cadenas, números o tuplas; si una tupla contiene cualquier objeto mutable directa o indirectamente, no puede usarse como clave. No es posible usar listas como claves, ya que las listas pueden modificarse.

Creación y manipulación de diccionarios

Para crear un diccionario vacío y luego asignar los elementos, lo declaramos con un par de llaves vacías que abren y cierran, y luego asignamos los valores directamente a las claves:

sinonimos = {}

sinonimos [“bailar”] = [“danzar”]

sinonimos [“ver”] = [“advertir”]

sinonimos [“vivir”]] = [“ser”,”existir”]

sinonimos [“mirar”] = []

sinonimos [“saltar”] = [“brincar”]

Si se provee una clave que no está en el diccionario se presentará una falla. Una forma de evitar que falle es utilizando la función get, que retorna el valor None si la clave no está en el diccionario.

Para verificar si una clave se encuentra en el diccionario, es posible utilizar la función has\_key o la palabra reservada in.

if sinonimos.has\_key('saltar'):

print sinonimos['saltar'] # Imprime brincar

if sinonimos.has\_key('amar'):

print sinonimos['amar'] # No se ejecuta

Recorridos de diccionarios

Adicionalmente, es posible también, obtener los valores como tuplas donde el primer elemento es la clave y el segundo el valor.

for palabra, sinonimo in sinonimos.items():

print palabra, ":", sinonimo

Métodos o Funciones adicionales sobre diccionarios

A continuación listamos algunas de las operaciones adicionales más utilizadas sobre diccionarios:

| **Operación** | **Descripción** |
| --- | --- |
| **cmp (dict1,dict2)** | # Compara el número de elementos distintos que tienen los dos |
| **dict.keys()** | # Devuelve una lista con las claves del diccionario |
| **dict.values()** | # Devuelve una lista con los valores del diccionario |
| **dict.get(key, default=None)** | # Devuelve el valor del elemento con clave key. Sino devuelve default |
| **dict.setdefault(key, default=None)** | # Inserta un elemento en el diccionario clave:valor. Si la clave existe no lo inserta |
| **dict['key'] = 'value'** | # Insertamos un elemento en el diccionario con su clave:valor |
| **dict.pop('key',None)** | # Eliminamos el elemento del diccionario con clave key |
| **dict.copy()** | # Devuleve la copia de un diccionario dict2 = dict.copy() |
| **dict.clear()** | # Elimina todos los elementos de un diccionario |
| **dict.fromkeys(list, defaultValue)** | # Crea un nuevo diccionario poniendo como claves las que hay en la lista y los valores por defecto si se les pasa |
| **dict.has\_key(key)** | # Devuelve true si existe la clave. Sino devuelve false |
| **dict.items()** | # devuelve un lista de tuplas formadas por los pares clave:valor |
| **dict.update(dict2)** | # Añade los elementos de un diccionario a otro |

Felicitaciones, hemos visto el tipo de dato estructurado diccionario, el cual podemos considerar como un conjunto no ordenado de pares clave: valor, y relacionarlo con conceptos conocidos como un diccionario tradicional o una agenda personal. Sigue adelante ampliando tus conocimientos acerca de este interesante lenguaje de programación.

1. Ejemplo: Diccionarios

Lección 5: Archivos en Python