Modulo 2 – Tipos de datos e instrucciones básicas en Python

Lección 1 – Tipos de datos y acciones elementales

1. Video 1: Constantes y variables en seudocódigo

* Facil de entender
* General
* Traducirse en lenguaje de programación

Valores:

* Elementales o Atomicos -> no se puede dividir
* Estructurados a partir de otros elementales

3 maneras:

* Constante
* Variable – que denota un valor genérico de un conjunto (contenedor de valores)
* Expresión – Otros valores con operadores (formulas o combinación de valores)

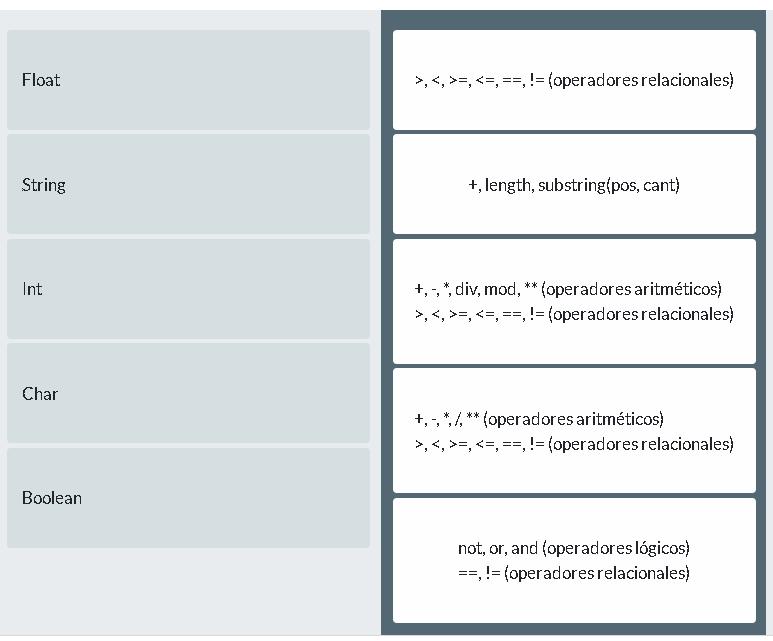
Tipos de datos:

* Tipos estructurados -Char (caracteres)
* Ordinal (mayor que o menor que …)
* String (Char) – operaciones
* Int – Valores numéricos positivos o negativos enteros
* Números reales: float
* Boolean: únicamente dos valores true y false (not, or y and)

1. Actividad 1

ANALIZA Y  
RELACIONA

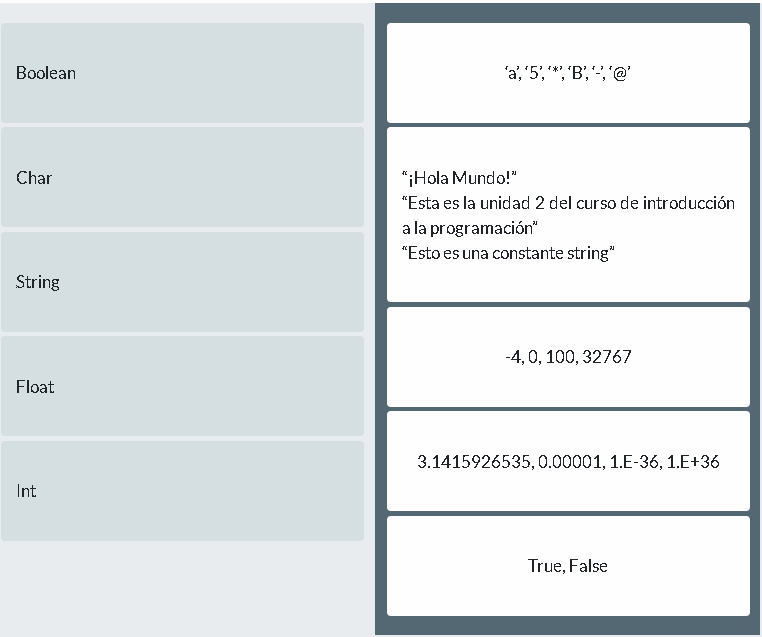
Relaciona cada tipo de dato con su respectiva opción de operaciones. Para ello, arrastra las opciones de izquierda a derecha según corresponda.



1. Actividad 2

ANALIZA Y  
RELACIONA

Relaciona cada tipo de dato con su respectiva opción de ejemplos. Para ello, arrastra las opciones de izquierda a derecha según corresponda



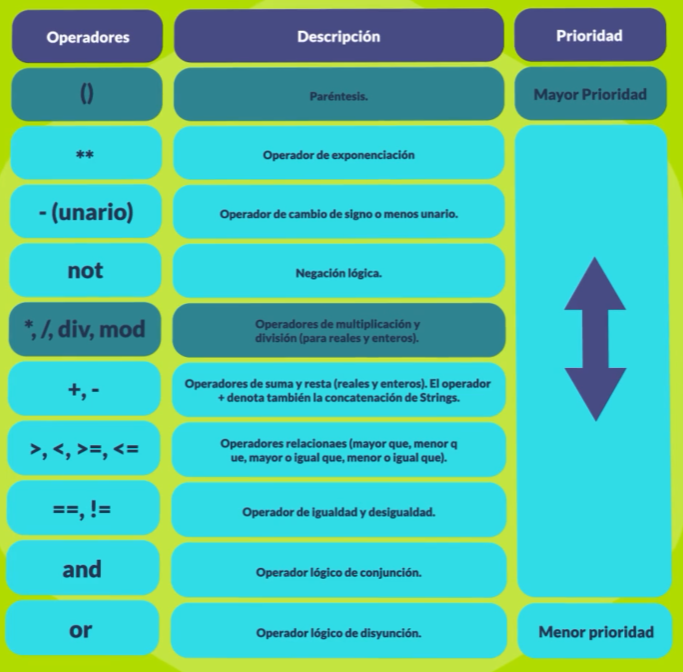
1. Video 2: Acciones Elementales

Declaraciones de variables, un tipo de datos seguido del nombre de la variable.

Fuertemente tipados y tienen la característica de que en tiempo de compilación se puede verificar que el uso de variables y expresiones respeta los distintos tipos.

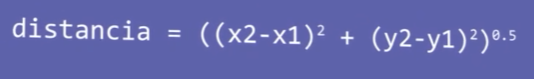
Python, Ruby y PERL no tipados (por lo general los lenguajes interpretados)

Regla de prioridad de operadores



Mecanismos de ingreso de datos

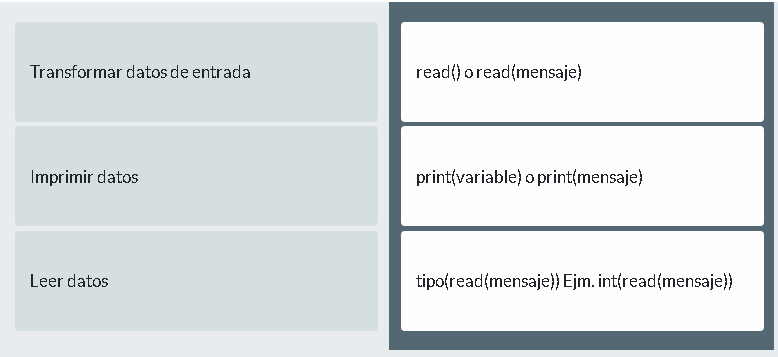
Ejemplo de Algoritmo:



1. Actividad 3

ANALIZA Y  
RELACIONA

¡Excelente noticia que ya comenzaste a conocer el pseudocódigo! Ahora contamos con un lenguaje que permite expresar de manera general soluciones computacionales, las cuales luego se pueden traducir a uno o más lenguajes de programación específicos. A continuación, te ofrecemos dos columnas que contienen la forma de realizar la lectura, impresión y transformación de datos en lenguaje pseudocódigo, te solicitamos relacionar la acción de la columna de la izquierda con su correspondiente sentencia en pseudocódigo que aparece en la columna de la derecha:



1. Ejercicio 1: Declaración de Variables
2. Actividad 4

Anímate a leer el texto y escoge la opción correcta según lo estudiado hasta ahora.

Afirmación: Un lenguaje poco tipado es

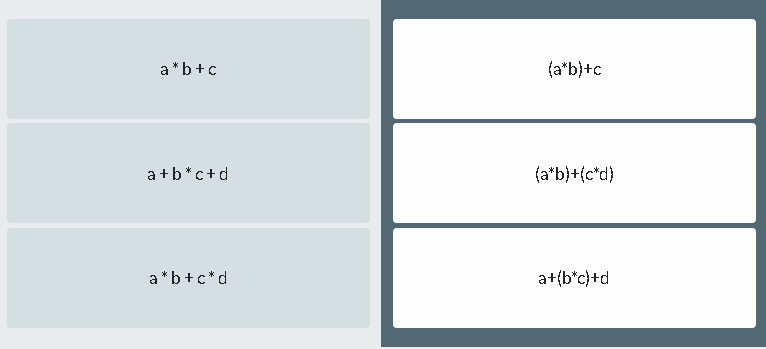
...aquel que...

Razón: En los lenguajes poco tipado el tipo de una variable puede cambiar en el tiempo e inclusive puede prescindir de éste en la declaración y se deduce en función de los valores que tome la variable. Por ejemplo, Python.

1. Ejercicio 2: Precedencia de operadores
2. Actividad 5

ANALIZA Y  
RELACIONA

¡Hola! Te damos la bienvenida a aplicar lo visto en relación a expresiones y operadores presentes en nuestro pseudocódigo. A continuación, te ofrecemos dos columnas que contienen expresiones matemáticas, te invitamos a relacionar la expresión sin paréntesis de la columna de la izquierda con su correspondiente expresión con paréntesis que aparece en la columna de la derecha:



Lección 2 – Variables y expresiones en Python

1. Video 1: Tipos Básicos

Int, float, texto

Enteros y float 🡪 Hexadecimal u octal

Booleam 🡪 True y False

1. Actividad 1

PRUEBA TUS  
CONOCIMIENTOS

¡Felicitaciones! Excelente idea poner en práctica lo estudiado con respecto a los tipos primitivos de Python, entre los cuales están los números enteros. Con base en esos conocimientos, indica cuáles de las siguientes representaciones de números enteros son formas válidas en Python:

1. Tipos Básicos en Python

CÁPSULA DE  
CONOCIMIENTO

Tipos básicos en Python

A continuación, te presentamos un resumen de los tipos básicos en Python, a saber: los tipos numéricos (float, int), texto (str) y booleanos (bool). Para cada uno te mostraremos sus aspectos básicos, operaciones y ejemplos de expresiones evaluadas en el intérprete de Python 3.

## **1.Tipos numéricos (int, float)**

En Python se tienen básicamente dos tipos numéricos: enteros **(int)** y reales **(float)**. Python soporta también números complejos **(complex numbers)**; sin embargo, no son considerados en este curso. Te recomendamos revisar la documentación oficial para mayores detalles.

A partir de Python 3 los enteros pueden tener tamaño ilimitado. De ser necesario, los valores enteros se pueden representar en hexadecimal u octal. Por ejemplo: 0xAF denota el valor entero 175 en hexadecimal y 0o100 denota el valor 64 en octal.

Los números reales se escriben separando mediante un punto la parte entera y la parte fraccional (por ejemplo, 3.1416) e inclusive pueden escribirse en notación científica, utilizando una tercera parte denotada por E, que indica la potencia de 10 asociada (por ejemplo, 2.5e2 = 2.5 x 102 = 250).

Python no ofrece sentencia de declaración de variables especificando el tipo asociado. En su lugar, cada vez que deseemos crear una variable debemos escribir instrucciones de la forma **<variable> = <expresión>**, en cuyo caso **<variable>** tendrá asociado el tipo de **<expresión>**. Por ejemplo, tenemos las siguientes definiciones de variables en Python:

|  |  |
| --- | --- |
| x = 6 | # x es una variable entera |
| hexa = 0xAF | # hexa es entero, inicializado en 175 |
| y = x+3 | # y es un entero, inicializado en el valor actual de x (6) más 3. |
| pi = 3.1415926535 | # pi es una variable float |
| x = pi\*\*2 | # la variable x ahora es de tipo float y tiene el valor de pi al cuadrado |

### **Conversiones de tipo**

En ocasiones, es necesario realizar conversiones explícitas de valores a enteros o reales. Para eso disponemos de las siguientes funciones:

* int(x): convierte el valor x a un entero (int)
* float(y): convierte el valor y a un real (float)

Python soporta aritmética mixta; es decir, cuando un operador aritmético tiene operandos de distintos tipos numéricos, el operando con el tipo más reducido es convertido al tipo más amplio. En consecuencia, cuando mezclamos operandos enteros y reales, el entero es convertido a real. Las comparaciones entre números de distintos tipos utilizan la misma regla.

### **Operaciones**

La siguiente tabla resume las operaciones soportadas por los tipos numéricos, están ordenadas de menor a mayor prioridad.

| **Operación** | **Observaciones** |
| --- | --- |
| **x + y** | Suma de x e y |
| **x - y** | Diferencia de x e y |
| **x \* y** | Producto de x e y |
| **x / y** | División de x e y |
| **x // y** | División entera (div) de x e y |
| **x % y** | Resto de la división entera (mod) de x e y |
| **-x** | x negado (cambio de signo) |
| **+x** | x sin cambios (identidad) |
| **abs()** | Valor absoluto de x |
| **int(x)** | x convertido a entero |
| **float(x)** | x convertido a real |
| **divmod(x,y)** | Retorna el par (x // y, x % y) |
| **pow(x,y)** | x elevado a la potencia y (función) |
| **x \*\* y** | x elevado a la potencia y (operador) |
|  |  |

## **2. Tipo secuencia de texto (str)**

Los valores textuales en Python son manejados con objetos de la clase str, o simplemente strings. Las constantes strings se pueden crear encerrando caracteres entre comillas simples o dobles. Para crear y manipular un objeto string se pueden asignar a variables, por ejemplo:

* var1 = '¡Hola Mundo!'
* var2 = "Curso de programación en ‘Python’"
* var3 = '''Tres comillas simples'''
* var4 = """Tres comillas dobles"""

Los strings formados por triples comillas pueden ocupar más de una línea.

En Python no existe el tipo caracter (char), por lo tanto, es considerado un string de tamaño 1 (o substring). Tanto para obtener elementos del string como substrings se utilizan corchetes. A manera de ejemplo, el código:

var1 = "Curso de programación en ‘Python’"

print("var1[0]: ", var1[0])

print("var1[25,33]: ",var1[25:33])

Imprime:  
**var1[0]: C var1[25,33]: ‘Python’**

### **Operaciones**

En la siguiente tabla se resumen algunos operadores de strings. En los ejemplos, la variable a contiene el string ‘Hola’ y la variable b contiene el string ‘Python’.

| **Operación** | **Descripción** | **Ejemplo** |
| --- | --- | --- |
| **+** | Operador de concatenación | a + b retorna ‘HolaPython’ |
| **\*** | Operador de repetición. Crea nuevos strings concatenando múltiples copias del mismo string. | a \* 2 retorna ‘HolaHola’ |
| **[ ]** | Obtiene el carácter ubicado en el índice 1. Los índices comienzan en cero. | a[1] retorna ‘o’ |
| **[ : ]** | Sustring. Obtiene los caracteres del rango especificado. | a[1:4] retorna ola |
| **in** | Test de membresía. Retorna True si un carácter existe en el string. | ‘o’ in a retorna True |
| **not in** | Test de no membresía. Retorna True si un carácter no existe en el string. | ‘z’ not in a retorna True |

Algunas funciones importantes para la manipulación de strings se listan en la tabla a continuación.

| **Función** | **Descripción** |
| --- | --- |
| **str.find(sub[,start[,end]])** | Retorna el menor índice en el string tal que el substring sub es encontrado en el substring [start:end] del string original. Los parámetros start y end son opcionales. La función retorna -1 si no encuentra a sub en el string. |
| **str.isallnum()** | Retorna True si todos los caracteres de un string son alfanuméricos y al menos uno es alfabético. |
| **str.isalpha()** | Retorna True si todos los caracteres del string son alfabéticos y hay al menos un caracter. En caso contrario, retorna False. |
| **str.isdigit()** | Retorna True si todos los caracteres del string son dígitos y hay al menos un caracter. En caso contrario, retorna False. |
| **str.islower()** | Retorna True si todos los caracteres del string son letras minúsculas y hay al menos un caracter. En caso contrario, retorna False. |
| **str.isupper()** | Retorna True si todos los caracteres del string son letras mayúsculas y hay al menos un caracter. En caso contrario, retorna False. |
| **str.len()** | Retorna la longitud o tamaño del string. |
| **str.lower()** | Retorna una copia del string con todos los caracteres convertidos a minúsculas. |
| **str.replace(old,new[,count])** | Retorna una copia del string con todas las ocurrencias de old reemplazadas por new. Si el argumento count es especificado, solo se reemplazarán las primeras count ocurrencias. |
| **str.split(sep=None,maxsplit=-1)** | Retorna una lista con todas las palabras del string, usando sep como delimitador. Si se indica un valor para maxsplit, se retornarán a lo sumo maxsplit particiones en palabras. |
| **str.upper()** | Retorna una copia del string con todos los caracteres convertidos a mayúsculas. |
| **str(x)** | Convierte x a string. |

## **3.Tipo booleano (bool)**

En Python los valores booleanos (tipo bool) son True y False; sin embargo, cualquier valor puede ser utilizado como un valor de verdad tanto en estructuras de control (condicionales y ciclos, que veremos en la Unidad 3) o como operandos en operaciones booleanas (or, and y not, descritas abajo). En general, en Python todo objeto es considerado True a menos que en su tipo/clase exista un método \_\_bool\_\_() que lo evalúe en falso o una operación \_\_len\_\_() que retorne al evaluarla con el objeto. Estos son algunos objetos evaluados como False en Python:

* Las constantes **None** y **False**
* Cualquier constante numérica que valga cero (0, 0.0,…)
* Strings vacíos (‘’ o “”) o colecciones vacías (listas, tuplas, rangos, conjuntos).

### **Operaciones**

Las operaciones booleanas en Python respetan las reglas lógicas de la conjunción, disjunción y negación; sin embargo, están optimizadas para que en ejecución se evalúe sólo lo necesario.

La tabla presentada a continuación contiene los operadores booleanos de menor a mayor prioridad junto a una descripción de cómo se evalúan.

| **Operación** | **Descripción** |
| --- | --- |
| **x or y** | Si x es falso, el resultado es y, sino el resultado es x. |
| **x and y** | Si x es falso, entonces el resultado es x, sino y. |
| **not x** | Si x es falso, entonces el resultado es True, sino False. |

Los operadores booleanos tienen menor prioridad que los operadores no booleanos; por ejemplo, la expresión not a == b es interpretada como not (a == b).

En Python hay ocho operadores de comparación, todos con la misma prioridad (mayor que la de los operadores booleanos). En la siguiente tabla se resumen los operadores de comparación:

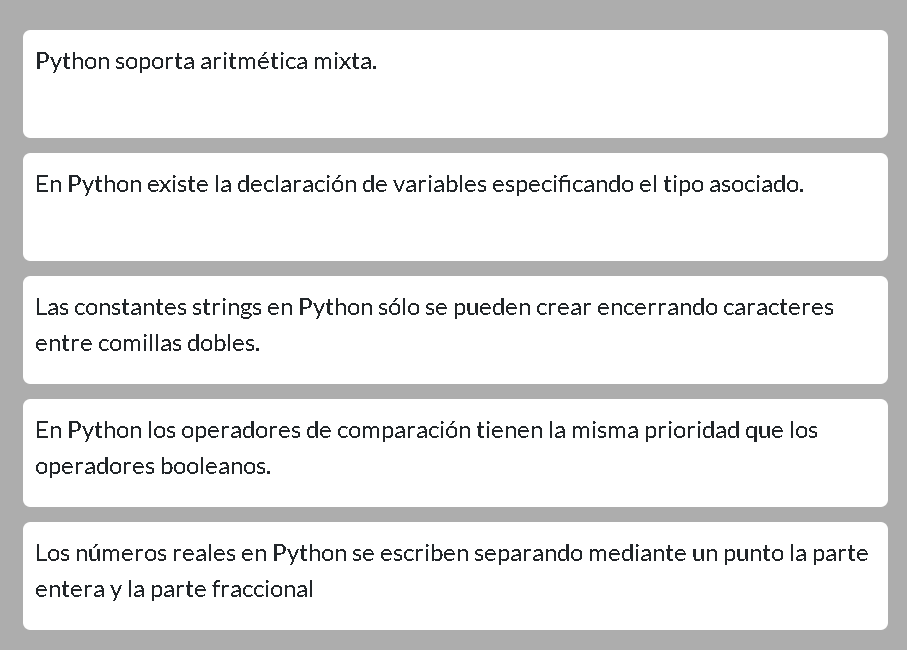
| **Operador** | **Significado** |
| --- | --- |
| **<** | Menor estricto que |
| **<=** | Menor o igual que |
| **>** | Mayor estricto que |
| **>=** | Mayor o igual que |
| **==** | Igual que |
| **!=** | Diferente que |
| **is** | Identidad de objetos |
| **is not** | Diferencia de objetos |

Hasta ahora has conocido los tipos básico de Python y las principales operaciones que puedes realizar con éstos, sigue adelante que cada vez se pone más interesante al ir profundizando tus conocimientos.

1. Actividad 2

PRUEBA TUS  
CONOCIMIENTOS

Ya hemos estudiado los tipos de datos elementales de Python, sus características y operaciones; considerando estos conceptos, indica cuáles de las siguientes afirmaciones son las correctas:



1. Ejercicio 1: Declaración de variables

DESAFÍO  
PRÁCTICO

Escribe un programa en Python que le permita al usuario conocer el monto en dólares americanos que tiene acumulado en su wallet o billetera digital correspondiente a una criptomoneda particular. En tal sentido, el programa debe leer el nombre de la criptomoneda, la cantidad acumulada de esa criptomoneda y su cotización por US$ del momento. Luego, debe calcular el monto total en US$ que posee el usuario e imprimir el siguiente mensaje: “Ud. posee un total de US$” seguido del cálculo realizado.

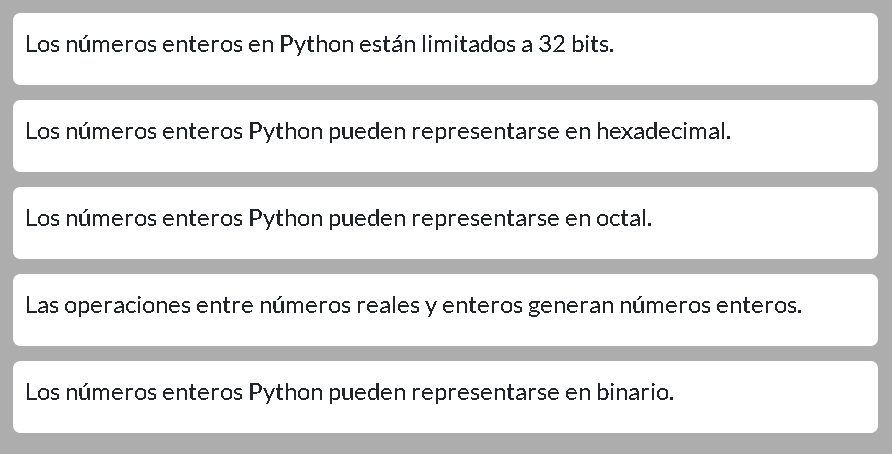
Recomendaciones:

* Asigna a las variables nombres nemotécnicos; es decir; que describan su contenido
* En Python no hay declaración de variables, se crean asignando el valor deseado

1. Actividad 3

PRUEBA TUS  
CONOCIMIENTOS

¡Saludos! Que bueno recibirte para consolidar lo que hemos estudiado acerca de los tipos de datos elementales de Python, sus características y operaciones; considerando estos conceptos, indica cuáles de las siguientes afirmaciones son las correctas:



Lección 3 – Instrucciones Básicas en Python

1. Video 1: Asignación

Tipo de las variables en Python es dinámico.

Asignaciones múltiples, iguales valores:

x = y = z = 0

Asignaciones múltiples, diferentes valores:

J, k, l = “Python3”, 3.14, 49

1. Asignación de expresiones y conversiones

CÁPSULA DE  
CONOCIMIENTO

Nombramiento de variables

El nombramiento de variables en Python sigue las reglas básicas de la mayoría de los lenguajes de programación:

* Los nombres de variables deben ser una única palabra (no puede haber espacios en blanco).
* Los nombres de variables se conforman únicamente de letras, números y guión bajo (underscore, \_ ).
* Los nombres de variables no pueden comenzar con un número.

En Python, las variables son sensibles a mayúsculas y minúsculas; por lo tanto, valor\_entero, VALOR\_ENTERO, Valor\_Entero y vAlor\_eNteRo son nombres de variables diferentes.

En cuanto al estilo para escribir variables, te recomendamos seguir la convención de comenzar con una letra minúscula y utilizar guion bajo para separar palabras. Comenzar con una letra mayúscula no es válido y algunos prefieren mezclar el uso de mayúsculas/minúsculas para separar palabras, pero no son más que posibles convenciones. Al final lo que importa es la consistencia.

En el documento de [estilo de codificación de Python](https://www.python.org/dev/peps/pep-0008/) (conocido como PEP 8, <https://www.python.org/dev/peps/pep-0008/> ) se describen las convenciones recomendadas para consistencia y legibilidad del código Python; te sugerimos que lo revises con detalle.

En Python hay un conjunto de nombres que no pueden ser utilizados para nombrar variables. Estos nombres son las denominadas palabras reservadas de Python y son utilizados para distintas construcciones del lenguaje. Las palabras reservadas de Python son: and, del, from, as, elif, global, assert, else, if, break, except, import, class, exec, in, continue, finally, is, def, for, lambda, not, while, or, with, pass, yield, print, raise, return, try.

Expresiones

Si tenemos expresiones como 2 + 3 \* 4 sabemos que existen reglas para determinar el orden en que se evalúan las operaciones. En este caso, se aplica primero el operador \* que posee mayor prioridad. En la primera lectura te mostramos las distintas operaciones para los tipos básicos de Python de acuerdo con el orden de prioridad. A continuación, te presentamos un resumen de las prioridades ordenadas ascendentemente en Python:

* if - else : expresión condicional (no confundir con la instrucción condicional)
* or : OR o disyunción booleana
* and : AND o conjunción booleana
* not x : negación booleana
* in, not in, is, is not, <, <=, >, >=, !=, == : operadores de comparación, incluyendo tests de membresía e – identidad
* | : OR a nivel de bits
* ^ : XOR a nivel de bits
* & : AND a nivel de bits
* <<, >> : operadores de desplazamiento a nivel de bits
* +, - : adición y sustracción
* \*, /, //, % : multiplicación, división, división entera y resto de la división entera
* +x, -x, ~x : signo positivo, negativo y negación a nivel de bits
* \*\* : operador de exponenciación
* x[índice], x[indice:indice], x(argumentos...), x.atributo : subscripción, subconjuntos, llamadas y referencia de atributos
* (expresiones...), [expresiones...], {clave: valor...}, {expressiones...} : asociación o despliegue de tuplas, listas, diccionarios y conjuntos

Es importante señalar que algunos operadores no se han mencionado aún, sin embargo se hará a lo largo del curso. En la lista se agrupan los operadores que tienen la misma prioridad.

Cuando se requiera hacer las expresiones más legibles o romper el orden de evaluación de los operadores se utilizan paréntesis para agrupar. Por ejemplo, la expresión mostrada anteriormente puede escribirse como (2 + 3) \* 4 para evitar que el operador de multiplicación se evalúe primero según la regla de prioridad.

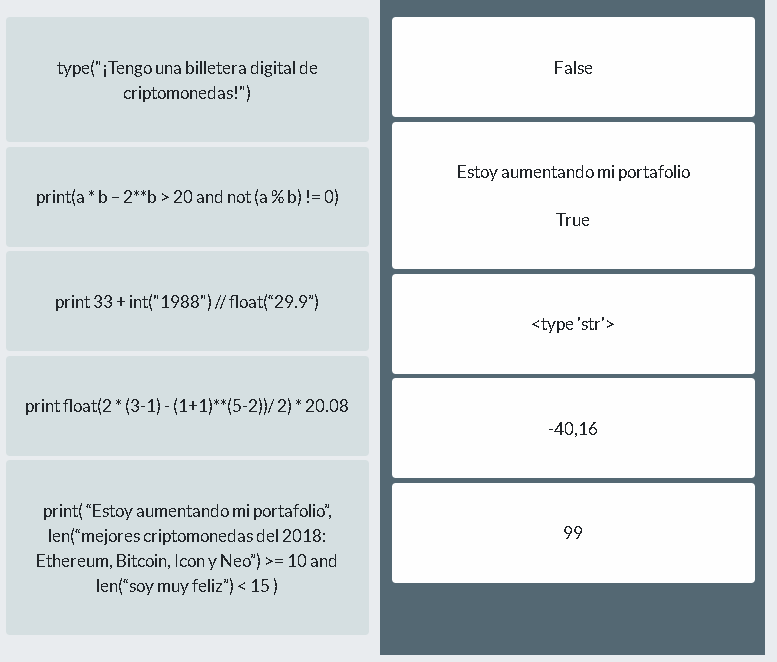
Los operadores se asocian de izquierda a derecha; esto quiere decir que cuando mezclamos operadores con igual prioridad, las expresiones se evalúan de izquierda a derecha. Por ejemplo: 2 + 3 + 4 se evalúa como (2 + 3) + 4.

1. Actividad 1

ANALIZA Y  
RELACIONA

¡Enhorabuena! Sigues progresando en tus conocimientos en Python, con esta actividad podrás consolidar lo referente al resultado de evaluar expresiones en Python teniendo en cuenta la prioridad de los diferentes operadores y conversiones de tipo.

A continuación, se te ofrecen dos columnas que contienen expresiones (columna izquierda) y los resultados de imprimir su evaluación; relaciona la expresión a imprimir de la columna de la izquierda con su correspondiente resultado que aparece en la columna de la derecha, considera a = 10 y b = 5.



1. Ejemplo 1: Definiciones de variables y tipos

Definición de valores y tipos de variables.

Convertir entre tipos

Convertir a str()

Convertir a entero int()

Una constante es una variable que no cambia en el tiempo

Para obtener el tipo type()

Variable sin valor: var = None

1. Video 2: Transformación de datos

Conversiones en Python:

Int()

Float()

Str()

Bool()

Type()

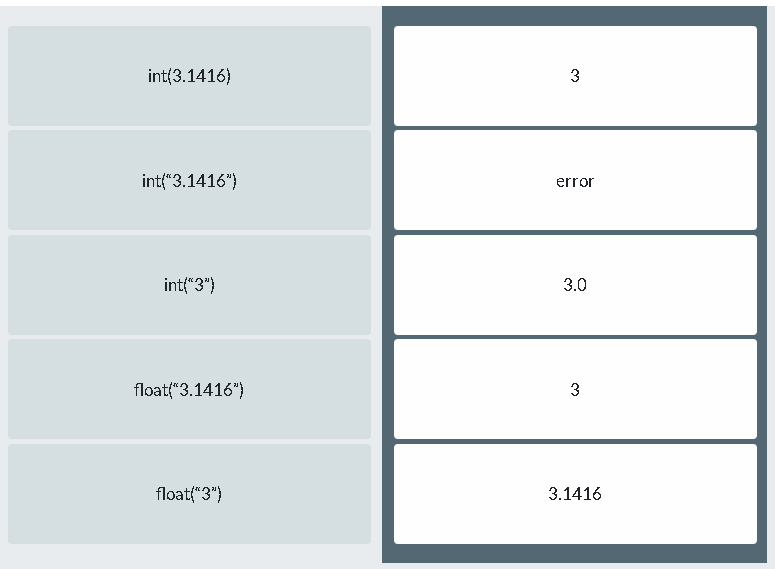
isinstance() 🡪 True or False

Ej: isinstance(num,float)

1. Actividad 2

ANALIZA Y  
RELACIONA

¡Hola! Estamos muy contentos que te dispongas a practicar lo visto acerca de la conversión de tipos de dato en Python. A continuación, te ofrecemos dos columnas que contienen transformaciones de tipo de datos en este lenguaje, relaciona la expresión de transformación de la columna de la izquierda con su correspondiente resultado que aparece en la columna de la derecha:



1. Ejemplo 2: Operaciones de lectura y escritura

Consola 🡪 Standard input 🡪 standard output

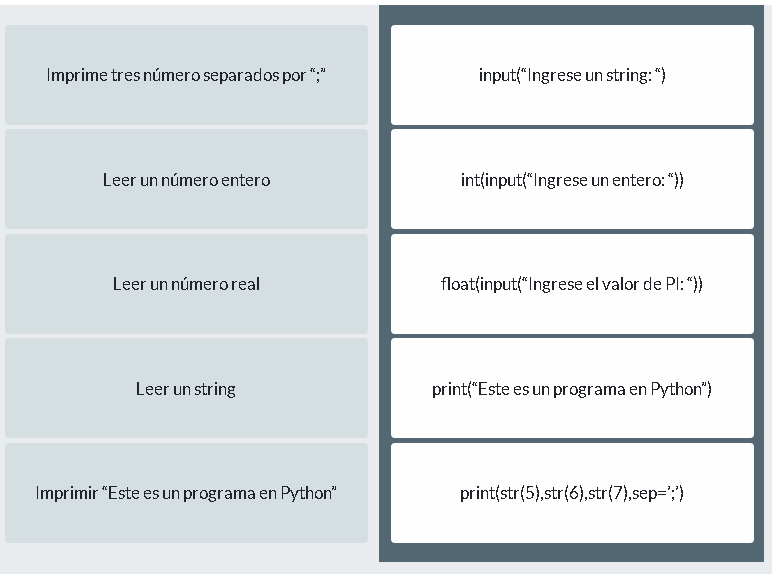
Input() 🡪 Espera por una entrada

Print(“Python”, 3, sep=’’) 🡪 El separador por defecto es espacio en blanco

1. Actividad 3

ANALIZA Y  
RELACIONA

¡Enhorabuena! Felicitaciones por tus avances con respecto a las operaciones básicas de lectura y escritura en el lenguaje de programación Python, aquí vamos a poner en práctica lo visto. A continuación, te ofrecemos dos columnas que contienen acciones a realizar en Python, relaciona la acción que se indica en la columna de la izquierda con su correspondiente sentencia en Python que aparece en la columna de la derecha:



1. Ejercicio 1: Lectura y Escritura

DESAFÍO  
PRÁCTICO

¡Felicitaciones por seguir avanzando! En esta oportunidad vamos a escribir un programa en Python, de esta forma emprendes el comienzo de un camino de soluciones que serán almacenadas en tu portafolio.

Te proponemos escribir un programa en Python que nos permita calcular la ganancia de una criptomoneda (Nombre y Cantidad), si ésta se negocia por una cantidad de días indicada por el usuario y a una ganancia fija indicada también por el usuario. Al terminar, el programa debe indicar la ganancia en cantidad de criptomoneda, la cantidad de días negociados y el monto total al finalizar los días.

Recomendaciones:

* Asigna a las variables nombres nemotécnicos; es decir; que describan su contenido
* En Python no hay declaración de variables, se crean asignando el valor deseado
* Hacer las lecturas y escrituras correspondientes.

Lección 4 – Funciones y módulos predeterminados en Python

1. Funciones predefinidas

CÁPSULA DE  
CONOCIMIENTO

Funciones predefinidas

Con esta lectura vas a seguir profundizando en los aspectos más importantes de Python que te permitirán conformar tu portafolio; en este sentido, vamos a destacar algunas funciones predefinidas en Python; además te presentamos cómo utilizar estas funciones desde un programa escrito en Python 3.

Funciones predefinidas

El intérprete de Python 3 posee un conjunto de funciones siempre disponibles para su uso. Estas funciones son denominadas funciones predefinidas (built-in functions) y son listadas a continuación con una breve descripción:

| **Función** | **Descripción** |
| --- | --- |
| **abs()** | Retorna el valor absoluto de un número |
| **all()** | Retorna true cuando todos los elementos en un iterable son true |
| **any()** | Verifica si algún elemento de un iterable es true |
| **ascii()** | Retorna un string que contiene una representación imprimible |
| **bin()** | Convierte un entero a un string binario |
| **bool()** | Convierte un valor a boolean |
| **bytearray()** | Retorna un nuevo arreglo de bytes |
| **bytes()** | Retorna un nuevo arreglo de bytes inmutable |
| **callable()** | Verifica si el objeto es invocable (callable) |
| **chr()** | Dado el código Unicode de un carácter, retorna la representación en string del mismo |
| **classmethod()** | Transforma una función en un método de clase |
| **compile()** | Retorna un código objeto que puede ser ejecutado con exec() |
| **complex()** | Crea un número complejo a partir de su parte real e imaginaria |
| **delattr()** | Elimina un atributo de un objeto |
| **dict()** | Crea un diccionario |
| **dir()** | Si no se especifican argumentos, retorna la lista de nombres en el alcance (scope) actual.  Si se indica un argumento, retorna la lista de atributos del objeto |
| **divmod()** | Retorna una tupla con la división entera y el resto de la división entera |
| **enumerate()** | Retorna un objeto enumerado (enumerate) |
| **eval()** | Evalúa una expresión con un ambiente global y local dados |
| **exec()** | Permite ejecutar código Python de manera dinámica |
| **filter()** | Construye un iterador con los elementos que hacen true a la función filtro |
| **float()** | Retorna un número real (float) a partir de un string o un número entero (int) |
| **format()** | Retorna una representación formateada de un valor |
| **frozenset()** | Retorna un objeto frozenset |
| **getattr()** | Retorna el valor de un atributo de un objeto |
| **globals()** | Retorna un diccionario con la tabla de símbolos globales actual |
| **hasattr()** | Retorna verdad si un objeto tiene un atributo con el nombre dado |
| **hash()** | Retorna el valor hash de un objeto |
| **help()** | En el modo interactivo, invoca al sistema de ayuda de Python |
| **hex()** | Convierte un entero (int) a hexadecimal |
| **id()** | Retorna el identificador único de un objeto |
| **input()** | Lee un valor y retorna un string |
| **int()** | Retorna un entero construido a partir de un string o un número real (float) |
| **isinstance()** | Verifica si un objeto es instancia de una clase dada |
| **issubclass()** | Verifica si un objeto es una subclase de una clase dada |
| **iter()** | Retorna un iterador para un objeto |
| **len()** | Retorna el tamaño/longitud de un objeto |
| **list()** | Crea una lista |
| **locals()** | Retorna un diccionario con la tabla de símbolos locales actual |
| **map()** | Retorna un iterador que aplica una función a cada elemento de un iterable dado |
| **max()** | Retorna el mayor elemento de un iterable o de una secuencia de dos o más argumentos |
| **memoryview()** | Retorna el objeto memory view asociado a un argumento |
| **min()** | Retorna el menor elemento de un iterable o de una secuencia de dos o más argumentos |
| **next()** | Obtiene el siguiente elemento de un iterador |
| **object()** | Crea un objeto de la clase base de todas las clases (object) |
| **oct()** | Convierte un entero (int) a octal |
| **open()** | Retorna un objeto archivo (file) |
| **ord()** | Retorna el código Unicode asociado a un caracter |
| **pow()** | Datos dos valores x y z, retorna el resultado de elevar x a la potencia z |
| **print()** | Imprime el objeto/valor dado |
| **property()** | Retorna un objeto propiedad (property) |
| **range()** | Dado un valor inicial y un valor final, retorna una secuencia de entero entre valor inicial y  final (ambos inclusive) |
| **repr()** | Retorna un string con una representación imprimible de un objeto |
| **reversed()** | Retorna un iterador reverso de la secuencia |
| **round()** | Redondea un número real (float) |
| **set()** | Retorna un objeto conjunto (set) a partir de los valores de una secuencia iterable |
| **setattr()** | Modifica el valor de un atributo de un objeto |
| **slice()** | Crea un objeto porción (slice) a partir de un rango |
| **sorted()** | Retorna una lista ordenada a partir de una secuencia iterable |
| **staticmethod()** | Crea un método estático a partir de una función |
| **str()** | Retorna la representación textual o en string de un objeto |
| **sum()** | Agrega elementos a una secuencia iterable |
| **super()** | Permite referenciar a la superclase |
| **tuple()** | Crea una tupla |
| **type()** | Retorna el tipo de un objeto o valor |
| **vars()** | Retorna el atributo \_\_dict\_\_ de una clase |
| **zip()** | Retorna un iterador de tuplas |

A continuación se presenta un ejemplo de uso de la función predefinida min() desde el prompt de Python 3:

>>> min(5,7,8,9,10,2)

2

Puedes incorporar todas estas funciones en tus programas, más adelante verás otros ejemplos de cómo utilizar algunas de éstas.

Luego de conocer algunas de las funciones predefinidas, te contamos que Python también ofrece una serie de módulos predefinidos que están a tu disposición y son descritos para tu conocimiento y estudio; así que sigue con este excelente ritmo de trabajo y descubrirás muchos otros temas interesantes de Python.

1. Actividad 1

ANALIZA Y  
RELACIONA

¡Saludos! Excelente momento para aplicar lo visto en relación a las funciones predefinidas en el lenguaje de programación Python. A continuación, te ofrecemos dos columnas que contienen información acerca de las funciones predefinidas de Python, te invitamos a relacionar la función de la columna de la izquierda con su correspondiente funcionalidad que aparece en la columna de la derecha:



1. Módulos predefinidos

CÁPSULA DE  
CONOCIMIENTO

Módulos predefinidos en Python

Al igual que las funciones predefinidas, Python ofrece una serie de módulos predefinidos que es importante que conozcas para continuar incrementando tus conocimientos y también tu portafolio. En tal sentido, te ofrecemos una lista de algunos módulos disponibles muy útiles.

Módulos Predefinidos

Además de las funciones predefinidas, Python provee un conjunto de módulos que soportan características muy útiles al momento de resolver problemas con Python. Un módulo contiene definiciones de funciones y opcionalmente puede contener instrucciones para inicializar el módulo. Un módulo es guardado como un archivo nombre\_modulo.py. Dentro del módulo se puede utilizar el nombre del módulo consultando el valor de la variable \_\_name\_\_.

Supongamos que poseemos un módulo con funciones y utilidades en Python para el manejo de figuras (figuras.py).

Para utilizar este módulo tanto desde el intérprete como desde otro módulo o programa se utiliza la sentencia: import figuras.

Esto produce que los nombres definidos dentro del módulo ahora estén disponibles usando el nombre del módulo. Por ejemplo, si se dispone de una función para calcular el área (area()) y tenemos una variable rect que contiene la información de una figura a la que queremos calcular el área se escribe:arect = figuras.area(rect) # se asigna en arect el resultado del cálculo del área de rect.

De ser necesario, se pueden importar elementos específicos de un módulo. Por ejemplo: from figuras import area, dibujar # importa únicamente las funciones área y dibujar.

Esto no requiere que se use el nombre del módulo para nominar el elemento a usar; en consecuencia, la llamada anterior se escribe como: arect = area(rect)

En caso de querer importar todos los elementos se utiliza \* en la lista de elementos a importar.

A continuación, se listan algunos de los módulos disponibles en Python agrupados según su utilidad:

| **Módulos** | **Utilidad** |
| --- | --- |
| **os, shutil, glob, sys** | Interfaz con el sistema operativo y el sistema de archivos |
| **re** | Manejo de expresiones regulares |
| **math** | Funciones matemáticas. Es importante destacar que este módulo  siempre está disponible, por lo cual para usarlo solo hay que usar  import math. |
| **urllib, smtplib, poplib, email** | Módulos para funcionalidades de diversos protocolos de internet |
| **datetime, timeit** | Manejo de fecha y hora |
| **zipfile, tarfile** | Compresión de datos |
| **doctest** | Módulo para ejecución de pruebas unitarias provistas en la  documentación del código |
| **xmlrpc** | Manejo de llamadas a procedimientos remotos |
| **json, csv, xml** | Manejo de formatos de intercambio de datos |
| **gettext, locale, codecs** | Internacionalización |
| **sqlite, dbm** | Manejo de bases de datos |
| **reprlib, pprint, textwrap** | Formateo de salida |
| **threading** | Soporte de multi-threading |
| **logging** | Soporte de logs (bitácoras) |

Además de estos módulos listados hay muchos módulos generados por la comunidad Python y disponibles con licencias de código abierto y disponibles para instalar desde [PyPi](https://pypi.python.org/pypi) (Python Package Index,).

Luego de conocer algunas de la funciones predefinidas y módulos disponibles en la comunidad de Python, te invitamos a seguir con las actividades del curso para que consolides tus conocimientos y enriquezcas tu portafolio.

1. Ejemplo 1: Módulos predefinidos

Date() 🡪 solo tiene la fecha

Datetime() 🡪 Tiene fecha y hora

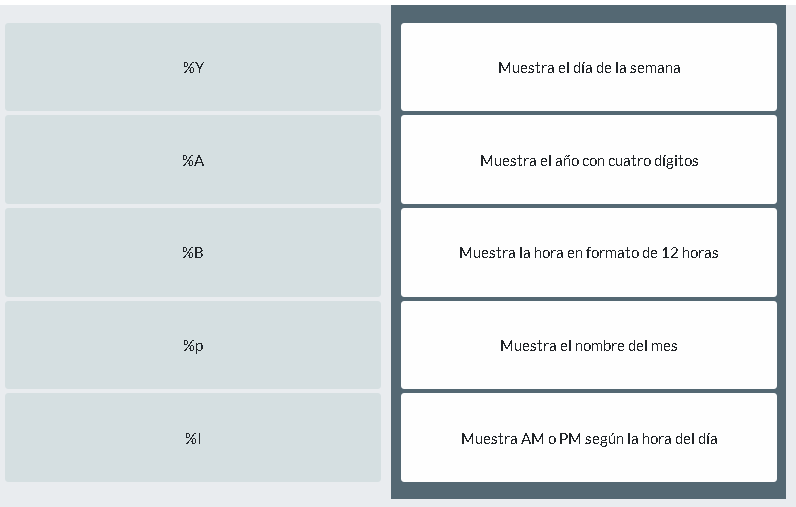
dt = datetime.now()

dt.strftime(“%d/%m/%y”) 🡪 Para dar formato de fecha y hora

1. Actividad 2

ANALIZA Y  
RELACIONA

¡Hola! Una vez más estamos muy felices que te dispongas a practicar lo visto acerca de los módulos predefinidos en el lenguaje de programación Python. A continuación, te ofrecemos dos columnas que contienen información del módulo datetime de Python, relaciona la formato de la columna de la izquierda con su correspondiente funcionalidad que aparece en la columna de la derecha:



1. Ejercicio 1: Funciones y módulos

DESAFÍO  
PRÁCTICO

¡Qué bien que vas a seguir incrementando tu portafolio! En este ejercicio vamos a aplicar los conocimientos adquiridos hasta el momento, haciendo énfasis en el uso de funciones y módulos predefinidos en Python.

Escribe un programa en Python que, dada una criptomoneda, la cantidad acumulada y su correspondiente cotización por dólar del día, le permita al usuario conocer la fecha completa y hora del momento en que obtuvo el sistema esa información.

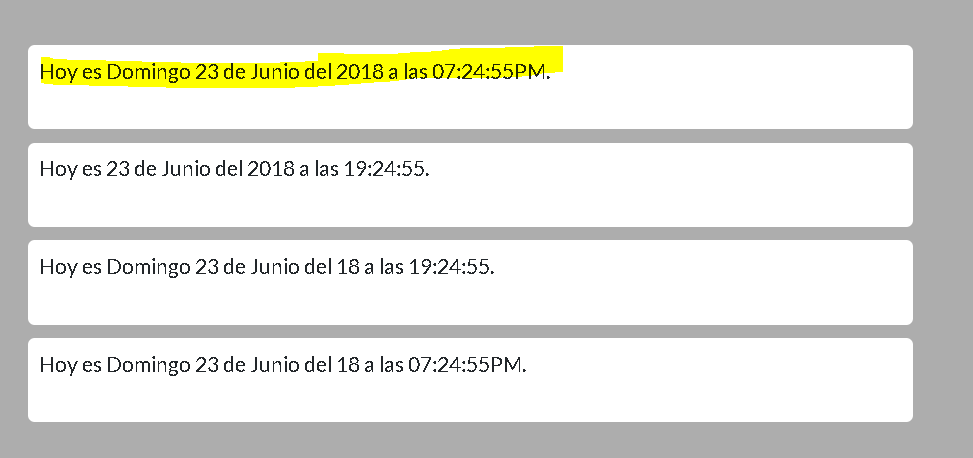
Recomendaciones:

* Considera el módulo datetime de Python
* Recuerda las operaciones de lectura y escritura
* Escribe los resultados en un formato adecuado para el usuario

1. Actividad 3

PRUEBA TUS  
CONOCIMIENTOS

¡Saludos! Es muy gratificante para nosotros que inviertas tiempo en aplicar lo estudiado acerca de los módulos predefinidos de Python, específicamente lo que hemos revisado en detalle del módulo de fecha y hora;con base en estos conceptos y tomando en cuenta el siguiente formato “Hoy es %A %d de %B del %Y a las %I:%M:S%p”, indica cuál es la salida correcta:



1. Ejercicio 2: Funciones y módulos

DESAFÍO  
PRÁCTICO

¡Qué bien que va a seguir creciendo tu portafolio! En este ejercicio vamos a aplicar los conocimientos adquiridos hasta el momento, haciendo énfasis en el uso de funciones y módulos predefinidos en Python.

Escribe un programa en Python que dada una criptomoneda, la cantidad acumulada y su correspondiente cotización por dólar del día, le permita al usuario conocer la fecha completa y hora del momento en que obtuvo el sistema esa información, así como el monto en US$ que tiene el usuario en su billetera digital. Considerando un crecimiento del 5% por día, muéstrale al usuario para ese mismo día de la siguiente semana cuál sería su ganancia en US$.

Recomendaciones:

* Considera el módulo datetime de Python
* Recuerda las operaciones de lectura y escritura
* Escribe los resultados en un formato adecuado para el usuario

1. Actividad 4

ANALIZA Y  
RELACIONA

¡Hola! Una vez más estamos muy felices que te dispongas a practicar lo visto acerca de los módulos predefinidos en el lenguaje de programación Python. A continuación, te ofrecemos dos columnas que contienen información del módulo datetime de Python, relaciona la formato de la columna de la izquierda con su correspondiente funcionalidad que aparece en la columna de la derecha:

