Python.industria(4.0)

PY004

Fundamentos de programación con Python – Parte II

Indice

X

Indice:

1. 01 Listas
2. 02 Diccionarios
3. 03 Tuplas
4. 04 Control de flujo
    - Condicionales
    - Bucles
    - La función Range
    - ~~Break y Continue~~
    - ~~Comprensión de listas~~

Listas

X

```
01(
    Listas
)
```

Listas [

En Python, las listas son una estructura de datos que permite almacenar y organizar múltiples elementos en una secuencia ordenada.

Las listas se definen utilizando corchetes [ ] y separando los elementos con comas. Es posible almacenar elementos de cualquier tipo en una lista, o **incluso combinar tipos**:

```
# Ejemplos de listas
numeros = [1, 2, 3, 4, 5]
planetas = ['Mercurio', 'Venus', 'Tierra', 'Marte', 'Júpiter', 'Saturno', 'Urano', 'Neptuno']
```

Las listas son **mutables**, lo que significa que se pueden agregar, eliminar y modificar elementos después de crear la lista.

]

Listas [

Ah! Además podemos tener una **lista hecha de listas**:

```
sensores_temperatura = [["Sensor1", "Sala de máquinas", 25.6],["Sensor2",
"Área de producción", 28.3],["Sensor3", "Almacén de productos", 22.1]]
```

Recordemos que una de las ventajas de Python es que **no presta atención a los espacios**, por lo cual podemos reestructurar nuestros datos para facilitar nuestro trabajo...

```
sensores_temperatura = [
    ["Sensor1", "Sala de máquinas", 25.6],
    ["Sensor2", "Área de producción", 28.3],
    ["Sensor3", "Almacén de productos", 22.1]
]
```

Mucho mejor, no?

]

Listas [

Para acceder a cada elemento dentro de las listas, basta con escribir el nombre de la lista, y la posición entre corchetes del dato. A esto se le llama **Indexado**.

Atención: Python usa índices en **base 0**

```
planetas[0]
```

‘Mercurio’

De la misma manera, si tenemos listas anidadas, podemos acceder a cada elemento poniendo los corchetes seguidos:

```
sensores_temperatura[1][2]
```

‘28.3’

```
sensores_temperatura = [
    ["Sensor1", "Sala de máquinas", 25.6],     #0
    ["Sensor2", "Área de producción", 28.3],   #1
    ["Sensor3", "Almacén de productos", 22.1]  #2
]
```

]

Listas [

Algo muy interesante del indexado en Python es que también funciona con números negativos.

¿Cuál es el planeta más alejado del sol?

```
planetas[-1]
```

‘Neptuno’

```
planetas[-2]
```

‘Urano’

]

Listas [

Otra cosa que podemos hacer con las listas es **Slicing**. Ésta técnica nos permite seleccionar **varios elementos** consecutivos, pertenecientes a una lista. Para ello utilizaremos el carácter especial **:** en conjunción a los índices para seleccionar.

Veamos 3 casos distintos

Caso 1:
Si escribo [0:5] seleccionaré de mi lista los elementos del **0** al **4**
Siempre se toma el índice de inicio hasta el índice del final, **pero sin incluir éste último.**

```
planetas [0:5]
```

```
['Mercurio', 'Venus', 'Tierra', 'Marte', 'Júpiter']
```

]

Listas [

Otra cosa que podemos hacer con las listas es **Slicing**. Ésta técnica nos permite seleccionar **varios elementos** consecutivos, pertenecientes a una lista. Para ello utilizaremos el carácter especial **:** en conjunción a los índices para seleccionar.

Veamos 3 casos distintos

Caso 2:
Si no escribo el índice del principio → [:3], se toma por default **el índice 0**. De esta manera puedo leer el campo como TODO, hasta el 3

```
planetas[:3]
```

```
['Mercurio', 'Venus', 'Tierra']
```

]

Listas [

Otra cosa que podemos hacer con las listas es **Slicing**. Ésta técnica nos permite seleccionar **varios elementos** consecutivos, pertenecientes a una lista. Para ello utilizaremos el carácter especial **:** en conjunción a los índices para seleccionar.

Veamos 3 casos distintos

Caso 3:
Si en cambio el que no escribo es el índice del final → [3:], se toma por default **el largo completo de la lista**. De esta manera puedo leer el campo como desde el 3, TODO

```
planetas[3:]
```

```
['Marte', 'Júpiter', 'Saturno', 'Urano', 'Neptuno']
```

]

Listas.modificaciones [

Como vimos anteriormente, las listas son “mutables”, lo que significa que podemos hacer cualquier cambio que queramos en el medio de un programa.

Por ejemplo, podríamos cambiar la temperatura de uno de los sensores asignándole el valor directamente a la expresión del índice:

```
sensores_temperatura = [
    ["Sensor1", "Sala de máquinas", 25.6],    #0
    ["Sensor2", "Área de producción", 28.3],   #1
    ["Sensor3", "Almacén de productos", 22.1]  #2
]

sensores_temperatura[2][2] = 20.0
sensores_temperatura
```

```
[['Sensor1', 'Sala de máquinas', 25.6],
['Sensor2', 'Área de producción', 28.3],
['Sensor3', 'Almacén de productos', 20.0]]
```

]

Listas.modificaciones [

Ejercicio:

Con las siguientes listas hacer las siguientes operaciones:

```python
# Ejemplos de listas
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]
planetas = ['Mercurio', 'Venus', 'Tierra', 'Marte', 'Júpiter', 'Saturno', 'Urano', 'Neptuno']
```

1- Cambiar los 3 primeros números a 0
2- Acortar el nombre de los últimos 3 planetas a las primeras 2 letras. ***Pista:*** *puedo asignar listas a una lista*
3- ¿Qué sucede si sumo **las dos listas entre sí?**

]

Listas.funciones() [

Las listas tienen funciones cómodas con las cuales podremos trabajar más eficientemente.

`len` nos da la longitud de una lista

```
numeros = [8,4,6,2,0,1,9,3,5,7]
len(numeros)
```

```
10
```

`sorted` nos da una versión ordenada de la lista

```
sorted(numeros)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

`sum` nos hace la sumatoria de todos los elementos de la lista

```
sum(numeros)
```

```
45
```

]

Listas.funciones() [

También, al ser objetos, tienen algunas funciones internas asociadas con las cuales podemos hacer otras tareas. Para acceder a éstas funciones internas debemos escribir nuestra lista, un ., y el nombre de la función

append agrega elementos al final de la lista

```
numeros = [8,4,6,2,0,1,9,3,5,7]
numeros.append(55)
```

```
[8, 4, 6, 2, 0, 1, 9, 3, 5, 7, 55]
```

pop elimina y nos devuelve el último elemento de la lista

```
ultimo_numero = numeros.pop()
numeros, ultimo_numero
```

Otras funciones útiles:

insert(pos,elemento) permite agregar un elemento a la posición deseada de la lista
del(índices) elimina elementos en la posición indicada
remove(elemento) elimina la primera aparición del elemento indicado en la lista

]

Listas.busqueda() [

¿Cómo podemos encontrar la posición de un elemento particular en una lista?

En qué posición de la cinta está la caja?

```
cinta = ["vacio","vacio","vacio","vacio","vacio","vacio","vacio","caja","vacio","vacio","vacio"]
```

Para ello utilizaremos la función interna index

```
cinta.index("caja")
```

```
7
```

¿Y dónde está la caja grande?

```
cinta.index("cajaGrande")
```

```
---------------------------------------------------------------------------
ValueError
Traceback (most recent call last) Untitled-1.ipynb
Cell 6 in 2 1 cinta =
["vacio","vacio","vacio","vacio","vacio","vacio","vacio","caja","vacio","vacio","vacio"] ----> 2
cinta.index("cajaGrande")

ValueError: 'cajaGrande' is not in list
```

]

Listas.busqueda() [

Para ahorrarnos éstas sorpresas, podemos en su lugar preguntar si la lista contiene cierto valor con el operador in.

```
cinta = ["vacio","vacio","vacio","vacio","vacio","vacio","vacio","caja","vacio","vacio","vacio"]
"caja" in cinta
```

True

```
"cajaGrande" in cinta
```

False

]

Listas.Ejercicios[

Dada una lista de temperaturas registradas en diferentes momentos del día, encontrar la temperatura mínima, máxima y promedio.

```
temperaturas = [26.75431352056954, 21.991086464852872, 27.209759257701366, 23.443668324912598, 28.8995217962772, 20.731695645337763,
22.58962536117254, 24.846285051127114, 26.074481791289337, 22.085936681499903, 23.913576519730877, 27.678759112104154,
24.381354548084965, 23.19239923228099, 20.803472013144523, 21.30390306534694, 27.102859960400624, 23.763191774421054,
25.157825319899203, 20.2290281340311, 26.746902019682647, 22.750408458130667, 20.26298637657635, 26.423788201931604]
```

Si la primera medición fue a las 00 hs, a qué horas fueron las máximas y las mínimas?

Dada una lista que contiene el estado (correcto o error) de múltiples sensores, contar la cantidad de sensores en estado de error. True = Error, False = Todo OK

```
estadoSensores = [True, True, True, False, False, False, True, False, True, False, False, True,
True, False, False, True, False, False, False, True]
```

]

Python.industria(4.0)

PY004

Fundamentos de programación con Python – Parte II

Volvemos en 5’

```
02(
    Diccionarios
)
```

Diccionarios {

Los diccionarios son una estructura de datos construida en Python, para mapear etiquetas (keys) únicas a valores específicos

Para definir diccionarios, usaremos llaves {}

```
sensores = {"Temperatura":0, "Presion":1, "Caudal":2}
```

En éste caso, aquellos valores encerrados entre comillas son las etiquetas, y los números son los valores

Para acceder a cada valor, usaremos los corchetes al igual que en las listas, solamente que en lugar de apuntar con un índice usaremos nuestra etiqueta

```
sensores["Presion"]
```

```
1
```

}

Diccionarios {

Si queremos añadir pares de valores a un diccionario, no hace falta declararlos de antemano. Simplemente escribimos la nueva etiqueta y le asignamos un valor

```
sensores["Nivel"] = 3
sensores
```

```
{'Temperatura': 0, 'Presion': 1, 'Caudal': 2, 'Nivel': 3}
```

Si queremos mostrar los valores o las etiquetas de un diccionario, basta con usar las funciones keys() o values() respectivamente.

```
sensores.keys()
```

```
dict_keys(['Temperatura', 'Presion', 'Caudal', 'Nivel'])
```

```
sensores.values()
```

```
dict_values([0, 1, 2, 3])
```

}

Diccionarios {

Ejercicio:

Volviendo a la lista de sensores en falla:

```
estadoSensores = [True, True, True, False, False, False, True, False, True, False, False, True,
True, False, False, True, False, False, False, True]
```

Armar un diccionario que tenga la cantidad de sensores en falla y sensores OK.

Finalmente, armar un print que nos muestre en forma textual “Hay ... Sensores ok y ... Sensores en falla”

}

Tuplas

X

```
03(
    Tuplas
)
```

Python.tuplas =

Las tuplas son casi exactamente igual a las listas, con dos diferencias fundamentales:

La sintaxis para crearlas usa paréntesis (o ningún elemento en absoluto) para crearlas:

```
tupla1 = (1,2,3)
tupla2 = 4,5,6
```

Por otro lado, y muy importante, son inmutables. ¡No podemos cambiar su valor una vez definidas!

```
tupla1(1) = 2
```

TypeError: 'tuple' object does not support item assignment

Python.tuplas =

Para qué se utilizan? Su uso más común es usarlas para funciones que retornan más de un valor.

Por ejemplo, existe la función as_integer_ratio() que me devuelve en forma de tupla el numerador y el denominador aproximados de un número con decimal

```
flot = 0.75
flot.as_integer_ratio()
```

```
(3, 4)
```

de esta manera, podemos guardar los valores cómodamente en dos variables separadas.

```
numerador, denominador = flot.as_integer_ratio()
```

O incluso invertir valores entre sí!

```
a = 1
b = 0
a, b = b, a
print(a, b)
```

Funciones_com

04(

Funciones comunes

Condicionales
Bucles
La función Range
~~Break y Continue~~
~~Comprensión de listas~~

)

Python.if:

En la clase anterior, vimos cómo podemos combinar múltiples condiciones lógicas entre sí, para tomar decisiones. Estas condiciones son especialmente útiles cuando se combinan con sentencias condicionales, o sentencias if-else

Estas sentencias permiten controlar, en función de una condición que se cumple, si una porción del código se ejecutará o no

Aquí tenemos una estructura básica, en donde podemos observar la sintaxis correcta:

```
caudal = 50.0

if caudal > 40.0:
    print("Sensor en falla")
else:
    print("Sensor OK")
```

Python.if:

Si tenemos varias condiciones que deben cumplirse, podemos usar la sentencia elif seguida de la condición para evaluar múltiples puntos a la vez

```
tanque = 85.7
if tanque > 95.0:
    estadoTanque = "Muy lleno"
elif tanque > 20.0:
    estadoTanque = "Operacion normal"
else:
    estadoTanque = "Vacio"

estadoTanque
```

```
'Operacion normal'
```

Python.for:

Una MUY poderosa herramienta de Python es la forma de utilizar bucles para recorrer grandes porciones de información y operar con ellas.

Una de las más comunes es el ciclo for, que ejecuta una operación una cantidad de veces determinada. Veamos un ejemplo

```
planetas = ['Mercurio', 'Venus', 'Tierra', 'Marte', 'Júpiter', 'Saturno', 'Urano', 'Neptuno']
for planeta in planetas:
    print(planeta)
```

```
Mercurio
Venus
Tierra
Marte
Júpiter
Saturno
Urano
Neptuno
```

Python.for:

Interesante, verdad? No tenemos que generar un índice, si no que le escribimos literalmente a Python qué es lo que queremos hacer:

“por cada planeta en la lista planetas, imprimí en la consola el planeta seleccionado”

```
planetas = ['Mercurio', 'Venus', 'Tierra', 'Marte', 'Júpiter', 'Saturno', 'Urano', 'Neptuno']
for planeta in planetas:
    print(planeta)
```

Incluso podemos hacer un loop de cada caracter en un String:

```
palabra = "caudalímetro"
lista_letras = []
for letra in palabra:
    lista_letras.append(letra)
```

Python.for:

Si queremos repetir una cantidad de veces determinada un bucle, podemos apoyarnos en la función “range”

range() es una función que devuelve una secuencia de números. Si queremos repetir una acción 7 veces, podemos usarla en conjunción con in.

```
for repeticion in range(7):
    print("Número de repetición: ", repeticion+1)
```

```
Número de repetición: 1
Número de repetición: 2
Número de repetición: 3
Número de repetición: 4
Número de repetición: 5
Número de repetición: 6
Número de repetición: 7
```

Python.A_programar!

Ejercicios:

Fácil - Monitoreo de Temperatura:

Verificar si las temperaturas registradas superan un umbral predefinido y mostrar las temperaturas anormales encontradas.

```
temperaturas = [28, 29, 31, 27, 33, 29, 30, 31, 32, 28]
```

Umbral: 30 grados Celsius
Utilizar un bucle for para recorrer la lista de temperaturas y una instrucción if para verificar si cada temperatura supera el umbral.
Crear una lista llamada temperaturas_anormales para almacenar las temperaturas que superen el umbral.
Al finalizar el bucle, imprimir las temperaturas anormales encontradas.

Python.A_programar!

Ejercicios:

Intermedio - Control de Stock:

Verificar si los productos tienen cantidades inferiores al punto de reposición y mostrar los productos con cantidades bajas.

```
productos_stock = {'producto1': 15, 'producto2': 7, 'producto3': 11, 'producto4': 5}
```

Umbral: 10 unidades
Utilizar un bucle for para recorrer las claves y los valores del diccionario y una instrucción if para verificar si alguna cantidad es inferior al punto de reposición.
Imprimir los productos con cantidades bajas y sus cantidades actuales, en un texto que sea amigable para el operario.