Python: resumen

viernes, 18 de marzo de 2022 19:25

Jerarquía conceptual en Python

Programas >> modulos >> statements >> expressions >> objetos

Las expresiones crear y procesan objetos
Objectcs también conocidos como data structures

Variable

Case-sensitive

Comentarios

Se usa (#) para hacer comentarios

Operadores lógicos

- Conjunción **and**
- Disyunción **or**
- Negación **not**

Nivel de precedencia:
not >> and >> or

Operadores matemáticos

Las operaciones (**+**) y (**-**) son asociativas de izquierda a derecha.
También ocurre con (*****) y (**/**)

Ejemplo: 3*4/2 => (3*4)/2

Operadores de identidad

Comprueba si ambos operandos hacen referencia al mismo objeto. Devuelven True o False

- **Is**
- **Is not**

Operadores de pertenencia

Comprueba si un valor o variable se encuentra en una secuencia(list, tuple, dict, str). Devuelven True o False

- **In**
- **Not in**

Nivel de precedencia

1. Paréntesis
2. Exponentes
3. División y multiplicación
4. Suma y Resta

Otros operadores:

- División entera (//)
- Resto (%)
- Exponenciación (**)

Tipo de variables

Int:
Representa a un número entero

Float:
Número con parte entera y parte fraccionaria

String (str)
Representar texto. Se puede usar comillas simples **' '** o dobles **" "**

Boolean (bool)
Puede tomar el valor de True o False

Lista (list)
Colección de valores, que pueden ser de cualquier tipo: int, float, string, incluso lista

Se usa corchetes **[ ]** y se separa cada valor con comas

Función type( )
Determina el tipo de variable

```
>> x=4.5
>> type(x)          # Output: float
```

Convertir el tipo de variable

- str( ): Para convertir a string
- int( ): Para convertir a entero
- float( ): Para convertir a flotante
- bool( ): Para convertir a booleano

Palabras claves

and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, nonlocal, None, not, or, pass, raise, return, True, try, with, while y yield.

Precedencia y operadores

Operación	Operador	Precedencia
Exponenciación	**	1
Multiplicación División División entera Módulo (resto)	* / // %	2
Suma Resta	+ -	3
Igual que Distinto a Menor que Menor o igual que Mayor que Mayor o igual que	== != < <= > >=	4
Negación	not	5
Conjunción	and	6
Disyunción	or	7

Asignación compuesta

Operador	Ejemplo	Equivalencia
+=	x+=1	x=x+1
-=	x-=1	x=x-1
*=	x*=2	x=x*2
/=	x/=2	x=x/2
%=	x%=2	x=x%2
**=	x**=2	x=x**2

Help

Se puede acceder a la documentación de alguna función, keyword, etc. a través de help o anteponiendo el signo de interrogación (?)

Ejemplo:

```
help(max)
?max
```

- str( ): Para convertir a string
- int( ): Para convertir a entero
- float( ): Para convertir a flotante
- bool( ): Para convertir a booleano
- list( ): Para convertir a lista

STRING - Cadena de caracteres

Concatenación (+)

1. Se puede concatenar dos o más strings con el operador **+** y dar como resultado un nuevo string
2. Se puede usar **%**, al lado se agrega una letra que indica el tipo de variable

```
>> nombre="Amelia"
>> edad = 17
>> r="Mi nombre es %s y tengo %i años" % (nombre, edad)
>>print(r)
   "Mi nombre es Amelia y tengo 17 años"
```

Tipo de variables

- %s -> str
- %i -> int
- %f -> flotante
- %.2f -> flotante con número especifico de decimales

F string

3. Se coloca la letra **f" "** antes de abrir comillas para el texto y se coloca el nombre de las variables entre corchetes **{var}**

```
>> nombre="Amelia"
>> edad = 18
>> r=f" Me llamo {nombre} y tengo {edad} años "
>>print(r)
   "Me llamo Amelia y tengo 18 años"
```

Métodos

Son funciones que pertenecen a un objeto (lista, string, float)

Métodos de str

Verificar el contenido del string

str.isalpha( )
Si contiene **sólo letras**.
Devuelve True o False

str.isdigit( )
Si contiene **sólo dígitos**.
Devuelve True o False

str.islower( )
Si contiene **sólo minúsculas**.
Devuelve True o False

str.isupper( )
Si contiene **sólo mayúsculas**.
Devuelve True o False

str.lower( )
Devuelve todo en minúsculas

str.upper( )
Devuelve todo en mayúsculas

str.startswith(value)
Devuelve **True** si el string **empieza** con el valor especificado

str.endswith(value)
Devuelve **True** si el string **termina** con el valor especificado

Modificar el contenido del string

str.find(value)
Devuelve la posición en la que encuentra **value**
Retorna -1 si no encuentra el valor

str.count(value)
Devuelve el número de veces que se **repite un carácter**

str.replace(old_value, new_value, count)
Reemplaza todos los old_value por los new_value. Devuelve una copia temporal.
- **Count**: El número de veces que se reemplazará. Opcional

str.strip(characters)
Elimina los espacios adelante y al final de la cadena
- **Characters:** caracteres que eliminar

str.split(separador)
Separa un string en una lista de acuerdo a un separador. El

Para string también se usa

[: : -1]
Se usa para invertir el orden de los caracteres de la cadena

```
>> frase="aloh"
>> inversa=frase[ : :-1]
>> print(inversa)  # hola
```

cadena.index("a")
Posición de un carácter dentro de una cadena

```
>> pos=frase.index("h")
>> print(pos)  # 3
```

separador por default es el espacio en blanco.

```
>> pos = lista.split("emma")
>> print(pos)
   2
```

separador.join(iterable)

Toma todos los ítems de un iterable y los junta en un string.

- **Iterable:** listas, tuplas, etc. Donde todos sus valores sean de tipo string
- El **separador** también debe ser tipo string

```
>> lista1 = ["Emma", "Liz", "John"]
>> sjunto= " ".join(lista1)
>> print(sjunto)
   Emma Liz John
```

str.partition("char")

Busca el char y separa un string en 3 elementos:

1) Parte del string antes de char
2) El char
3) Parte del string después del char

Devuelve una tupla con los 3 elementos

```
>> saludo ="Hola! me llamo Z"
>> tupla = saludo.partition("!")
>> print(tupla)
   ("Hola", "!", " me llamo Z")
```

Busca la primera ocurrencia del char

ord(str)

Convierte un ascii a un entero

chr(int)

Convierte un entero a un ascii

LISTAS [ ]

Permiten modificar su contenido, son mutables
Para acceder a la información en la lista se usa el **índice**

Ejemplo:

```
  0   1       2        3    4      5     6    7
["liz", 1.73 , "emma", 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
lista= ["liz", 1.73 , "emma", 1.68, 'mom', 1.71, 'dad', 1.89]
lista[6]            # Output: 'dad'
```

También hay índices negativos, son útiles si se quiere obtener elementos del final de la lista

```
 -8   -7     -6       -5    -4     -3   -2    -1
["liz", 1.73 , "emma", 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
lista[-2]                # Output: 'dad'
```

Slicing

Selecciona múltiples elementos de una lista, creando una nueva lista. Se ingresa un rango **[start: end]**

- No se incluye el valor en la posición **end**
- El valor por default de **start** es **0**
- Si no se coloca un valor en **end** se asume que debe llegar **hasta el final**

Lista invertida [: : -1]

Invierte el orden de los valores de la lista

```
>> listaI=lista[ : :-1]
>> print(listaI)
   [1.89, 'dad', 1.71, 'mom', 1.68, 'emma', 1.73, 'liz']
```

Métodos de lista

Dir (object)

Built-in function que lista los nombres de los métodos disponibles para el objeto que se le ha pasa como argumento

random.shuffle(lst)

Función de la librería random, reordena el orden de los ítems de una lista (o secuencia)

lista.index(x , start, end)
Devuelve la posición en qué aparece un determinado elemento dentro de una lista

```
>> pos = lista.index("emma")
>> print(pos)
   2
```

Opcional: [start, end] indica en que porción de la lista de va a buscar es valor "x"

lista.count(x)
Retorna el número de veces que x aparece en la lista

lista.append(x)
Agregar un ítem al final de la lista

lista.insert(i, x)
Inserta un ítem en una posición dada.

- **i -->** índice del ítem delante del cual se insertará x
- **x -->** ítem que se insertará

lista.extend(iterable)
Extiende la lista agregando todos los ítems del iterable

```
>> lista1 = [1, 2]
>> lista2 = [3, 4, 5]

>> lista1.extend(lista2)
>> print( lista1 )
    [1, 2, 3, 4, 5]
```

lista.clear( )
Elimina **todos** los elementos de la lista

lista.pop(i)
Elimina el ítem en la posición i de la lista y lo retorna.
Cuando no se especifica el valor de i, se elimina y retorna el último elemento

lista.remove(x)
Elimina el primer ítem de la lista cuyo valor sea x

lista.reverse( )
Invierte el orden de los elementos de la lista

lista.sort(*reverse=False*)
Ordena los elementos de forma **ascendente** por default. Para usar este método todos los elementos deben ser del mismo tipo.
Ejemplo:

```
>> lista = [90, 56, 3, 45, 16, 34, 7, 45]
>> lista.sort( )
>> print( lista)
   [3, 7, 16, 34, 45, 45, 56, 90]

>> frutas = ["pera", "manzana", "fresa", "papaya"]
>> frutas.sort( )
>> print( frutas)
    ['fresa', 'manzana', 'papaya', 'pera']

>> lista2 = [90, 56, 3, 45, 16, 34, 7, 45]
>> lista2.sort( reverse= True)
>> print( lista2)
   [90, 56, 45, 45, 34, 16, 7, 3]
```

El parámetro Reverse por default está en False, cuando se usa **reverse = True** se ordena los elementos de forma **descendente**

Indicación **del**

Elimina un ítem de la lista usando su índice. También puede eliminar secciones de la lista si se le proporciona el rango

```
>> A= [90, 56, 3, 45, 16, 34, 7, 45]
>> del A[0]
>> print( A)
   [56, 3, 45, 16, 34, 7, 45]

>> del A[4 : len(A) ]
```

```
>> del A[4 : len(A) ]
>> print( A)
   [56, 3, 45, 16]
```

Además puede borrar variables

TUPLAS ( )

Son inmutables: no se puede adicionar, eliminar, reemplazar ni reordenar sus elementos
Puede contener objetos mutables como listas
También puede contener otra tupla

Crear tupla

```
tupla= (73 , "emma",  1.89, (3, "luz"))
tupla[3]              # Output: (3, "luz")
```

Desempaquetar tupla

```
tupla= (73 , "emma",  1.89, (3, "luz"))
a, b, c, d = tupla
d             # Output: (3, "luz")
```

CONJUNTOS { }

- Desordenado, no hay indexación.
- Estructura de datos no secuencial
- No puede contener listas
- Los elementos repetidos se eliminan

```
 # Crear conjunto vacío
>> conj = set( )
 # Crear conjunto
 >> conj = {1, 2, 3}
```

Métodos de conjuntos

conj.add(x)
Añade el elemento **x** al conjunto. Si ya existe no lo agrega

conj.clear( )
Borra todos los ítems del conjunto

conj.copy( )
Devuelve una copia del conjunto

conj.discard(x)
Borra el ítem **x** del conjunto

Comparación de conjuntos

conj.isdisjoint(conj2)
Devuelve **True** si no hay intersección entre conj y conj2

conj.issuperset(conj2)
Devuelve **True** si **conj > conj2**

conj.issuperset(conj2)
Devuelve **True** si **conj > conj2**

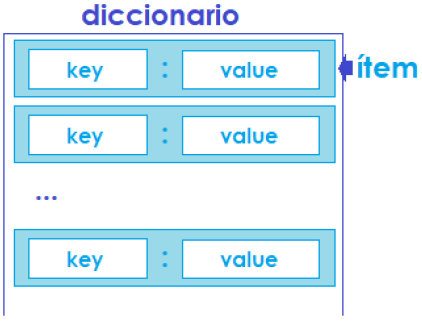
conj.update(conj2)
Agrega todos los elementos del conj2 en **conj**

conj.union(conj2)
Devuelve un nuevo conjunto de la unión de conj y conj2

conj.difference(conj2)
Devuelve **conj - conj2**

conj.intersection(conj2)
Devuelve la intersección entre conj y conj2

conj.symmetric_difference(conj2)
Devuelve los elementos que no se repiten en ambos conjuntos



DICCIONARIOS { }

DICCIONARIOS { }

Almacena una colección de pares clave/valor (key, value)
No hay claves duplicadas, debe ser inmutable
Cada clave corresponde a un valor. **Clave + valor = ítem**

```
>> diccionario = { }            #diccionario vacío
>> diccionario[key] = value          #Agregar un ítem
>> #Si la clave ya existe, el valor correspondiente será reemplazado
>> del diccionario[key]          #Eliminar un ítem
```

Obtener el value dando el key

- **dicc[key]** -> si no existe el key nos devuelve un error
- **dicc.get(key)** -> si no existe el key nos devuelve None

Métodos de diccionario

- ### Recorrer las claves

```
>> valores = {"A":4, "E": 3, "I": 1, "O": 0}
>> for k in valores:
        print( k)
>> A
>> E
>> I
>> O
```

- ### Iterar sobre los valores

```
>> valores = {"A":4, "E": 3, "I": 1, "O": 0}
>> for v in valores.values( ):
        print( v)
>> 4
>> 3
>> 1
>> 0
```

Iterar sobre clave y valor de cada elemento del diccionario

```
>> valores = {"A":4, "E": 3, "I": 1, "O": 0}
>> for k, v in valores.items( ):
        print( "k=", k, ", v=", v)
>> k=A, v=4
>> k=E, v=3
>> k=I, v=1
>> k=O, v=0
```

Nota: imprime el key y value a la vez

Funciones

- Palabra clave **def**
- Solo interpreta la función cuando la llama
- Importante la indexación
- Se utiliza **return** para devolver un valor, si no devuelve ningún valor el -predeterminado en None
- Las variables definidas dentro de una función solo existen dentro del ámbito de esa función

```
>> def NombreFunct(parametro1, parametro2):
      print ("Hello " +  parametro1 + "y " parametro2 )

   NombreFunct(Ana, Pedro)
   Hola Ana y Pedro

>> def duplica(num):
     x=num*2
     return x

>> print(x)   # error - X no está definida
   doble =duplica(4)
   8
```

Built-in functions

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s

Built-in functions

- **len(object):** Devuelve la longitud de un objeto o el número de ítems
- **max(iterable):** Devuelve el ítem más grande de un iterable o el más grande entre 2 parámetros como max(n1, n2, n3, ...)
- **min(iterable):** Devuelve el ítem con el valor más bajo. Si los valores son string se hace una comparación alfabética
- **sum(iterable):** Devuelve la suma de todos los ítems en un iterable
- **chr(97):** Recibe un entero y devuelve carácter
- **ord("a"):** Recibe un string y devuelve un entero
- **all(iterable):** Retorna True si todos los ítems del iterable son True
- **any(iterable):** Retorna True si algún ítem del iterable es True
- **input (prompt):**
  La función lee una línea de input, lo convierte a string y lo retorna.
  Prompt: Un string, el mensaje que va antes del input
- **enumerate (iterable, start):**
  Recibe un iterable y lo retorna como un objeto numerado. La función realiza un seguimiento de dos valores: el índice y el valor del elemento
  - **Start:** Un número, el default es 0

```
>> employee_names= ["Paul", "John", "Abbie", "Ana", "Ron"]
>>
>> for index, name in enumerate(employee_names, 1):
        print( index,name)

   1 Paul
   2 John
   3 Abbie
   4 Ana
   5  Ron
```

- **sorted(iterable, key, reverse=False):**
  Recibe un iterable y retorna una lista ordenada sin modificar el iterable original.
  - **Key:** Una función que servirá como base de comparación
  - **Reverse:** por default está en False, la lista se ordena de forma ascendente.

```
# Ordenar una lista basado en su resto de dividirlo entre 7
>> def func(x):
        return x % 7          # resto: 1, 3, 4, 0

>> L=[ 15, 3, 11, 7]
>> print( sorted(L, key=func) )

   [ 7, 15, 3, 11 ]
```

- **Open(file, mode='r')**
  Abre un archivo y retorna el objeto archivo correspondiente.
  - **File:** La dirección y el nombre del archivo
  - **Mode:** Para especificar el modo en el cual se abrirá el archivo

Carácter	Significado
' r '	Leer el archivo(default)
'w'	Escribir, remueve los contenidos del archivo sin borrar el archivo
' x '	Creación, falla si el archivo ya existe
' a '	Escribir, agregando al final del archivo, si existe
' b '	Modo binario
' t '	Modo texto (default)
' + '	Para actualizar (leer y escribir)

Función Lambda

48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Función Lambda

Es una función anónima que es definida sin un nombre. Para definir estas funciones se usa el keyword **lambda.** Pueden tener cualquier número de argumentos pero solo una expression. La expresión es evaluada y retornada

Sintaxis: **lamba arguments: expression**

```
>> double = lambda x: x*2
>> print (double(5))
   10
```

If y esle

```
>> if cond1:
>>     bloque de código
>> elif cond2:
>>     bloque de código
>> else:
>>     bloque  de código
```

While

```
>> while condición :
>>     bloque de código
```

Se puede alterar el flujo de ejecución con:

- **Break**
  Se utiliza para finalizar y salir del bucle
- **Continue**
  Salta el siguiente paso de la iteración, ignorando las sentencias que le siguen.

For

Recorre los elementos de un iterable y ejecuta un bloque de código.

- **Ítem:** variable que toma el valor de cada elemento dentro del iterador.

```
>> for <ítem> in <iterable>:
>>     bloque de código
```

- **range(start, stop, step)**
  Retorna una secuencia inmutable de números dentro de un determinado rango. Se puede usar de 2 formas:

  range(stop): 1 argumento, por default start=0
  range(start, stop, step): 2 argumentos y uno opcional [step]

```
>> for num in range(0, 8, 2):
>>     print(num)
    0
    2
    4
    6
```

For... else

Else en un bucle for se refiere al código que se ejecutará cuando el bucle termine. Se ejecutará siempre que no haya un break en el for

```
>> for <ítem> in <iterable>:
>>     bloque de código
>> else:
>>     bloque de código
```

Files

Lectura de archivos

Open(file, mode='r')

Abre un archivo y retorna el objeto archivo correspondiente

Procesador

Memoria RAM

Guarda datos de forma temporal
Cuando se ejecuta un programa en Python

- Se alamacena en la memoria RAM
- Cuando termine de ejecutar se borra

Disco duro

Es donde se guardan los archivos
Se almacenan en discos

- Ficheros
  - Binarios --> .bin ejem: 010101
  - Planos --> .txt cadenas de texto

Crear archivo

Creemos un archivo de texto:

```
[ ]  %%file lista_de_compras.txt
     arroz
```

Lectura de archivos

Open(file, mode='r')

Abre un archivo y retorna el objeto archivo correspondiente.

- **File:** La dirección y el nombre del archivo
- **Mode:** Para especificar el modo en el cual se abrirá el archivo

Carácter	Significado
' r '	Leer el archivo(default)
' x '	Creación, falla si el archivo ya existe
' + '	Para actualizar (leer y escribir)

Readline( )

Devuelve la primeda línea

```
ruta = "lista_de_compras.txt"
archivo= open (ruta, "r")
registro = archivo.readline()
archivo.close( )
```

Si se coloca uno despues de otro la 2da línea, 3era y así

Read( )

Devuelve todo (no solo la 1era línea) como string

```
registro = archivo.read()
```

Readlines( )

Devuelve todo como una lista
Cada línea es un elemento de la lista

```
%%file cursos.txt
Programación Orientada a Objetos, 15
Fisica 1, 20
Calculo 1, 18
Química, 25
```

```
[ ]  ruta = "cursos.txt"
     archivo = open(ruta, "r")
```

```
[ ]  registro = archivo.readlines()

for element in registro:
  print(element)

Programación Orientada a Objetos, 15

Fisica 1, 20

Calculo 1, 18

Química, 25


archivo.close()
```

Seek(offset, *whence*)

Establece el cursor del archivo en una posición específica

2 parámetros:

- Offset: set the cursos en una posición específica y empieza a leer después de dicha posición
- Whence: Opcional. Punto de referencia
  - **0:** Default. Inicio del archivo
  - **1:** Posición actual del archivo
  - **2:** Posición al final del archivo

```
[ ]  %%file lista_de_compras.txt
     arroz
     azucar
     leche
     pan

     Writing lista_de_compras.txt
```

```python
# Opening a file for reading
with open('sample.txt', 'r') as file:
    # Setting the cursor at 62nd position
    file.seek(62)
    # Reading the content after the 62nd character
    data = file.read()
    print(data)
```

Tell( )

Devuelve la posición del cursor del archivo. Sin parámetros

Escritura de archivos

Open(file, mode='r')

Abre un archivo y retorna el objeto archivo correspondiente.

- **File:** La dirección y el nombre del archivo
- **Mode:** Para especificar el modo en el cual se abrirá el archivo

Carácter	Significado
'w'	Escribir, remueve los contenidos del archivo sin borrar el archivo
' a '	Escribir, agregando al final del archivo, si existe
' + '	Para actualizar (leer y escribir)

Lectura y escritura

with open(ruta, 'r') as archivo1:

```python
ruta2 = "comprados.txt"
with open(ruta2, "w") as archivo1:
  archivo1.write("Pam")
archivo1.close()
```

Try / finally

```python
# forma correcta de hacer una lectura o escritura:
try:
  with open(ruta2, "w") as archivo1:
    archivo1.write("Pan")
finally:
  archivo1.close()
```

Error and Exceptions

Parsing