FACULTAD DE INGENIERÍA
UNICEN • OLAVARRÍA

Fundamentos de la Programación y Métodos Numéricos

Estructuras de Datos

Agenda – Cadenas, Listas, Tuplas y Diccionarios

- Cadenas
- Listas
- Tuplas
- Diccionarios
- Ejercicios

Agenda – Cadenas, Listas, Tuplas y Diccionarios

- Cadenas
- Listas
- Tuplas
- Diccionarios
- Ejercicios

Cadenas

Cadenas: Las cadenas representan un tipo de dato compuesto y son cuantitativamente diferentes de los tipos de datos **int**, **float**, **boolean**, etc. Podemos tratar un tipo compuesto como un único elemento e ir accediendo a cada una de sus partes. Ejemplo:

```
elemento = "cadenas"
letra = elemento[1]
print (letra) # imprime la letra a
```

La expresión **elemento[1]** selecciona el carácter numero 1 de **elemento**. A la expresión entre corchetes se le llama índice.

Notar que las cadenas comienzan en el índice 0.

```
letra = elemento[0]
print (letra) # imprime la letra c
```

Cadenas

Longitud de las cadenas: La función **len** devuelve el numero ´ de caracteres de una cadena. Ejemplo:

```
elemento = "cadenas"
print (len(elemento)) # imprime 7
```

Obtener la ultima letra:

```
longitud = len(elemento)
ultima = elemento[longitud] # IndexError: string index out of range.
```

Para obtener el ultimo caracter tenemos que restar 1 de longitud:

```
longitud = len(elemento)
ultima = elemento[longitud-1]
print (ultima)  # imprime s
```

Cadenas

Recorrido de las cadenas: Una forma de codificar un recorrido es una sentencia while:

```
elemento = "cadenas"
indice = 0
while indice < len(elemento):
        letra = elemento[indice]
        print (letra) # imprime c a d e n a s
        indice = indice + 1
```

Otra alternativa:

```
for letra in elemento:
        print (letra) # imprime c a d e n a s
```

Cadenas

Recorrido de las cadenas:

```
prefijos = "ABC"
sufijo = "abc"

for letra in prefijos:
        print (letra + sufijo)
```

La salida del programa es:

```
Aabc
Babc
Cabc
```

Cadenas

Porciones de las cadenas: Llamamos porción a un segmento de una cadena. La selección de una porción es similar a la selección de un carácter:

```
texto = "Pedro, Pablo, y Maria"
print (texto[0:5]) # imprime Pedro
print (texto[7:12]) # imprime Pablo
print (texto[16:21]) # imprime Maria
```

El operador **[n:m]** devuelve la parte de la cadena desde el enésimo carácter hasta el "enésimo", incluyendo el primero pero excluyendo el ultimo.

Cadenas

Porciones de las cadenas: Si omite el primer índice (antes de los dos puntos), la porción comienza al principio de la cadena. Si omite el segundo ´índice, la porción llega al final de la cadena. Por ejemplo:

```
texto = "cadenas"
print (texto[:3]) # imprime 'cad'
print (texto[3:]) # imprime 'enas'
```

Cadenas

Comparación de las cadenas: Los operadores de comparación trabajan sobre cadenas. Para ver si dos cadenas son iguales

```
texto = "cadenas"
if texto == "cadenas":
        print ("Si, es cadenas")
```

Otro ejemplo (las mayúsculas van antes de la minúsculas):

```
texto = "cadenas"
if texto < "Cz":
        print ("Menor")
elif texto > "Cz":
        print ("Mayor") # imprime Mayor
else:
        print ("Igual")
```

Cadenas

Función encuentra: Que hace la siguiente función ??

```
def encuentra(cadena, caracter):
        indice = 0
        while indice < len(cadena):
                if cadena[indice] == caracter:
                        return indice
                indice = indice + 1
        return -1
```

Cadenas

Función contar: Que hace la siguiente función ??

```
def contar(cadena):
        cuenta = 0
        for c in cadena:
                if c == 'a':
                        cuenta = cuenta + 1
        return cuenta
```

Cadenas

Modulo "str" : El modulo (str) **string** contiene funciones útiles para manipular cadenas.

El modulo **str** incluye una función llamada **find** que hace lo mismo que la función encuentra que escribimos.

```
texto = "cadenas"
indice = str.find(texto, "a")
print (indice)   # imprime 1

indice = str.find(texto, "de")
print (indice)   # imprime 2
```

Agenda – Cadenas, Listas, Tuplas y Diccionarios

- Cadenas
- Listas
- Tuplas
- Diccionarios
- Ejercicios

Listas

Listas: Una **lista** es un **conjunto ordenado de valores**, en el cual cada valor va identificado por un **índice**. Los valores que constituyen una lista son sus elementos. Las listas son similares a las **cadenas de texto** (**strings**), que son conjuntos ordenados de caracteres, excepto en que los elementos de una lista pueden ser de cualquier tipo. Ejemplo:

```
Lista1 = [10, 20, 30, 40]
Lista2 = ["spam", "email", "texto"]
```

Los elementos de una lista no tienen por que ser del mismo tipo.

```
Lista3 = ["hola", 2.0, 5, [10, 20]]
```

Una lista dentro de otra lista es una **lista anidada**.

Listas

Accesos a los elemento: La sintaxis para acceder a los elementos de una lista es la misma que para acceder a los caracteres de una cadena: el operador corchetes **[]**. La expresión dentro de los corchetes especifica el índice.

```
numeros = [10, 20, 30, 40]
print (numeros[0]) # [10]
numeros[1] = 5
print (numeros)  # [10, 5, 30, 40]
```

Se puede usar como índice cualquier expresión entera:

```
print (numeros[4-4]) # [10]
```

```
numeros[1.0] # TypeError: sequence index must be integer
numeros[20] = 5 # IndexError: list assignment index out of range
numeros[-1] = 5 # IndexError: list assignment index out of range
```

Listas

Longitud: La funcion **len** toma una lista y devuelve su tamaño. Ejemplo de uso:

```
numeros = [10, 20, 30, 40]
i =0
while i < len(numeros):
        print (numeros[i]) # 10, 20, 30, 40
        i = i + 1
```

Listas

Existe elemento en una lista: El operador **in** retorna un valor booleano que comprueba la pertenencia a una secuencia

```
numeros = [10, 20, 30, 40]

print (10 in numeros) # imprime 1

print (15 not in numeros) # imprime 1
```

Listas

Recorrido: El bucle **for** es más conciso porque podemos eliminar la variable de bucle, **i** (ver caso del uso del **while**). Aquí tenemos el bucle anterior con un bucle **for**:

```
numeros = [10, 20, 30, 40]
for n in numeros:
        print (n) # imprime 10 20 30 40
```

Se puede usar cualquier expresión de lista en un bucle **for**:

```
for numero in range(20):
        if numero % 2 == 0:
                print (numero) # imprime 0,..,18
```

Listas

Operaciones con listas:

El operador + concatena listas:

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print (c) # imprime [1, 2, 3, 4, 5, 6]
```

el operador * repite una lista un numero dado de veces:

```
print ([0] * 4) # imprime [0, 0, 0, 0]
print ([1, 2, 3] * 3) # imprime [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Listas

Porciones (slices): Las operaciones de porciones que vimos en cadenas también funcionan en sobre las listas:

```
lista = ['a', 'b', 'c', 'd', 'e', 'f']
print( lista[1:3]) # imprime['b', 'c']
print( lista[:4]) # imprime['a', 'b', 'c', 'd']
print( lista[3:]) # imprime['d', 'e', 'f']
print( lista[:]) # imprime ['a', 'b', 'c', 'd', 'e', 'f']
```

Borrado de un elemento: la function **del** elimina un elemento de una lista:

```
a = ['uno', 'dos', 'tres']
del a[1]
print (a) # ['uno', 'tres']
```

Listas

Clonar listas: Si queremos modificar una lista y mantener una copia del original necesitamos clonar una lista:

La forma más fácil de clonar una lista es por medio del operador de porción:

```
a = [1, 2, 3]
b = []
b[:] = a[:]
print (b) # imprime [1, 2, 3]
```

Listas

Listas anidadas: Una lista anidada es una lista que aparece como elemento dentro de otra lista. En esta lista, el tercer elemento es una lista anidada:

lista = ["hola", 2.0, 5, **[10, 20]**]

Si imprimimos **lista[3]**, obtendremos **[10, 20]**. Para extraer los elementos de la lista anidada, podemos proceder en dos pasos:

```
elemento = lista[3]
print (elemento[0]) # imprime 10
print (elemento[3][1]) # imprime 20
```

Listas

Listas anidadas: Una lista anidada es una lista que aparece como elemento dentro de otra lista. En esta lista, el tercer elemento es una lista anidada:

lista = ["hola", 2.0, 5, **[10, 20]**]

Si imprimimos **lista[3]**, obtendremos **[10, 20]**. Para extraer los elementos de la lista anidada, podemos proceder en dos pasos:

```
elemento = lista[3]
print (elemento[0]) # imprime 10
print (elemento[3][1]) # imprime 20
```

Listas

Matrices: Una matriz representan listas anidadas para representar matrices. Por ejemplo, la matriz

matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print (matriz[1]) **# imprime la fila 1 [4, 5, 6]**

Tomar un elemento de la matriz:

print (matriz[1][1]) **# imprime 5**

Agenda – Cadenas, Listas, Tuplas y Diccionarios

- Cadenas
- Listas
- Tuplas
- Diccionarios
- Ejercicios

Tuplas

Hemos visto dos tipos compuestos: **cadenas** (compuesta de caracteres) y **listas** (compuesta de elementos de cualquier tipo). Una de las diferencias entre listas y cadenas es que los elementos de una lista se pueden modificar, pero los caracteres de una cadena no. En otras palabras, las cadenas son **inmutables** y las listas son mutables.

Tupla: Una tupla es similar a una lista salvo en que es **inmutable**. Sintácticamente, una tupla es una lista de valores separados por comas:

```
tupla1 = 'a', 'b', 'c', 'd', 'e'
# o
tupla2 = ('a', 'b', 'c', 'd', 'e')
```

Tuplas

Creación de tuplas: Para crear una tupla con un solo elemento, debemos incluir una coma final:

```
t1 = ('a', 'b')
print( type(t1)) # <type 'tuple'>
```

Python trata ('a') como una cadena entre paréntesis:

```
t2 = ('a')
print (type(t2)) # <type 'string'>
```

Tuplas

Operaciones: las operaciones sobre las tuplas son las mismas que sobre las listas. El operador índice selecciona un elemento de la tupla:

```
tupla = ('a', 'b', 'c', 'd', 'e')
print (tupla[0]) # imprime 'a'
```

El operador de porción selecciona un intervalo de elementos.

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

Sli intentamos modificar uno de los elementos de la tupla provocaremos un error:

```
tupla[0] = 'A' # TypeError: object doesn't support item assignment
```

Tuplas

Al no poder modificar los elementos de una tupla, podemos sustituir una tupla por otra diferente:

```
tupla = ('A',) + tupla[1:]
print (tupla) # imprime ('A', 'b', 'c', 'd', 'e')
```

Tuplas

Asignación de tuplas: Para intercambiar los valores de dos variables necesitamos a y b:

```
temp = a
a = b
b = temp
```

Python proporciona otra alternativa para este problema:

```
a, b = b, a
```

El lado izquierdo es una tupla de variables, el lado derecho es una tupla de valores. Cada valor se asigna a su respectiva variable. Todas las expresiones del lado derecho se evalúan ´ antes de las asignaciones. Esta característica hace de la asignación de tuplas algo muy versátil.

Tuplas

Asignación de tuplas: Para intercambiar los valores de dos variables necesitamos a y b:

```
temp = a
a = b
b = temp
```

Python proporciona otra alternativa para este problema:

```
a, b = b, a
```

El lado izquierdo es una tupla de variables, el lado derecho es una tupla de valores. Cada valor se asigna a su respectiva variable. Naturalmente, el numero de variables a la izquierda y el numero de valores a la derecha deben ser iguales:

```
a, b, c, d = 1, 2, 3 # ValueError: unpack tuple of wrong size
```

Tuplas

Tuplas como valor de retorno: Las funciones pueden devolver tuplas como valor de retorno. Por ejemplo, podríamos escribir una función que intercambie dos parámetros:

```
def intercambio(x, y):
        return y, x
```

Luego podemos asignar el valor de retorno a una tupla con dos variables:

```
a, b = intercambio(a, b)
```

Agenda – Cadenas, Listas, Tuplas y Diccionarios

- Cadenas
- Listas
- Tuplas
- Diccionarios
- Ejercicios

Diccionarios

Los tipos compuestos que ha visto hasta ahora (cadenas, listas y tuplas) usan enteros como ´índices. Si intenta usar cualquier otro tipo como índice provocaría un error.

Los diccionarios son similares a otros tipos compuestos excepto en que pueden usar como ´índice cualquier tipo inmutable. A modo de ejemplo, crearemos un diccionario que traduzca palabras inglesas al español. En este diccionario, los índices son **strings** (cadenas).

Una forma de crear un diccionario es empezar con el diccionario vacío y añadir elementos. El diccionario vacío se expresa como {}:

```
ing_a_esp = {}
ing_a_esp['one'] = 'uno'
ing_a_esp['two'] = 'dos'
```

Diccionarios

Podemos presentar el valor actual del diccionario del modo habitual:

print (ing_a_esp) **# imprime {'one': 'uno', 'two': 'dos'}**

Los elementos de un diccionario aparecen en una lista separada por comas. Cada entrada contiene un índice y un valor separado por dos puntos **(:)**. En un diccionario, los índices se llaman **claves**, por eso los elementos se llaman pares **clave-valor**.

Otra forma de crear un diccionario es dando una lista de pares clave-valor con la misma sintaxis que la salida del ejemplo anterior:

```
ing_a_esp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
print(ing_a_esp) # imprime {'one': 'uno', 'three': 'tres', 'two': 'dos'}
print (ing_a_esp['two']) #imprime 'dos'
```

Diccionarios

Operaciones sobre diccionarios: La sentencia **del** elimina un par clave-valor de un diccionario. Por ejemplo, el diccionario siguiente contiene los nombres de varias stock de un comercio:

```
inventario = {'tornillos' : 430, 'arandelas' : 312, 'clavos' : 525}
print (inventario)
# imprime {'tornillos': 430, 'arandelas': 312, 'clavos': 5257}
```

Si alguien compra todas las peras, podemos eliminar la entrada del diccionario:

```
del inventario['clavos']
print (inventario) # imprime {'tornillos': 430, 'arandelas': 312}
```

Diccionarios

O si esperamos recibir más clavos pronto, podemos simplemente cambiar el inventario asociado con los clavos:

```
inventario['clavos'] = 0
print inventario # imprime {'tornillos':430, 'arandelas':312, 'clavos':0}
```

La función **len** también funciona con diccionarios; devuelve el numero de pares **clave-valor**:

```
print (len(inventario)) # imprime 4
```

Diccionarios

Métodos del diccionario:

El método **keys** acepta un diccionario y devuelve una lista con las claves que aparecen, pero en lugar de la sintaxis de la función **keys**(ing_a_esp), usamos la sintaxis del método **ing_a_esp.keys()**.

print (ing_a_esp.**keys**()) **# imprime ['one', 'two', 'three']**

El método **values** es similar; devuelve una lista de los valores del diccionario:

print (ing_a_esp.**values**()) **# imprime ['uno', 'tres', 'dos']**

Diccionarios

Métodos del diccionario:

El método **items** devuelve ambos, una lista de tuplas con los pares clave-valor del diccionario:

```
print (ing_a_esp.items()) # imprime [('one','uno'), ('three', 'tres'), ('two', 'dos')]
```

El método **has_key** acepta una clave y devuelve verdadero (1) si la clave aparece en el diccionario:

```
print (ing_a_esp.has_key('one')) # imprime 1
print ( ing_a_esp.has_key('deux')) # imprime 0
```

Agenda – Cadenas, Listas, Tuplas y Diccionarios

- Cadenas
- Listas
- Tuplas
- Diccionarios
- Ejercicios

Ejercicios:

Escriba en Python el siguiente programa:

1) Realizar una búsqueda de las funciones típicas que tiene el modulo ***string***

2) Realice la creación de los tipos de cadenas, listas, tuplas y diccionarios.

3) Para cada uno de los tipos vistos realice el alta, baja, modificación y consulta.