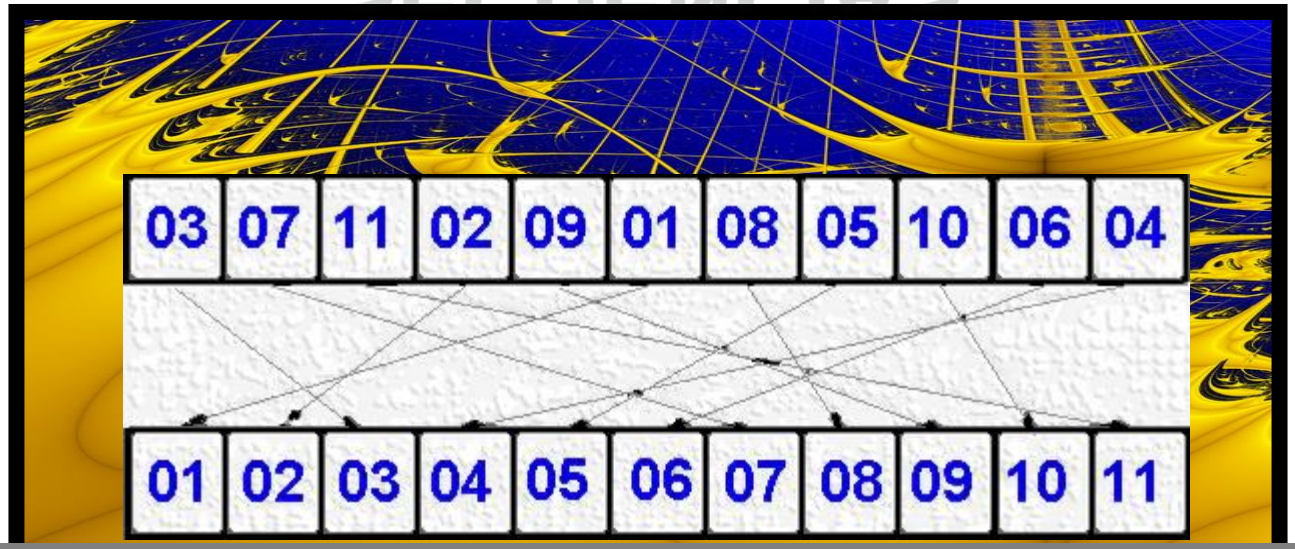
Cátedra de Informática – Dpto. Computación
Universidad Nacional de Córdoba.

Ing. Ventre, Luis O.

CAPITULO 5:

TIPOS ESTRUCTURADOS

SECUENCIAS



- Hasta el momento hemos utilizado 3 TIPOS de datos:

ENTEROS | FLOTANTES | CADENAS

- Los dos primeros, enteros y flotantes son tipos de datos escalares: UN UNICO ELEMENTO ATOMICO.

- Las cadenas, sucesión de caracteres, son un tipo de dato SECUENCIAL.

- Los datos secuenciales, son datos ESTRUCTURADOS.

TEMARIO

- 5.1 Cadenas
  - 5.1.1 Lo que ya sabemos
  - 5.1.2 Escapes
  - 5.1.3 Longitud de una cadena
  - 5.1.4 Indexación
  - 5.1.5 Recorrido de cadenas
  - 5.1.6 Un ejemplo: un contador de palabras
  - 5.1.7 Otro ejemplo: un prog. conversión binario a decimal.
  - 5.1.8 A vueltas con las cadenas, inversión de una.
  - 5.1.9 Subcadenas: el operador de corte
  - 5.1.10 Una aplicación: correo electrónico personalizado
  - 5.1.11 Referencias a cadenas

- 5.2 Listas
  - 5.2.1 Cosas que ya sabemos
  - 5.2.2 Comparación de listas
  - 5.2.3 El operador IS
  - 5.2.4 Modificación de elementos de Listas
  - 5.2.5 Mutabilidad, inmutabilidad y representación de la info
  - 5.2.6 Adición de elementos a una lista
  - 5.2.7 Lectura de listas por teclado
  - 5.2.8 Borrado de elementos de una lista
  - 5.2.9 Pertenencia de un elemento a una lista
  - 5.2.10 Ordenación de una lista

- 5.1 Cadenas
  - 5.1.1 Lo que ya sabemos

- 5.1.2 Escapes
- 5.1.3 Longitud de una cadena
- 5.1.4 Indexación
- 5.1.5 Recorrido de cadenas
- 5.1.6 Un ejemplo: un contador de palabras
- 5.1.7 Otro ejemplo: un prog. Conversión binario a decimal.
- 5.1.8 A vueltas con las cadenas, inversión de una.
- 5.1.9 Subcadenas: el operador de corte
- 5.1.10 Una aplicación: correo electrónico personalizado
- 5.1.11 Referencias a cadenas

- Def: Sucesión de caracteres encerradas entre ‘ ‘.
- Operadores:
  - Operador + (concatenación)
  - Operador * (repetición de cadenas)
  - Operador % (sustitución de marcas de formato)
  - int: recibe una cadena y devuelve el int que describe
  - float: idem float.
  - str: se le pasa un entero o float y convierte a cadena.
  - ord(‘a’) – chr (0-255)
- Manipulación:
  - a.lower ( ) a.capwords( )
  - a.upper( )

- 5.1 Cadenas
  - 5.1.1 Lo que ya sabemos
  - **5.1.2 Escapes**
  - 5.1.3 Longitud de una cadena
  - 5.1.4 Indexación
  - 5.1.5 Recorrido de cadenas
  - 5.1.6 Un ejemplo: un contador de palabras
  - 5.1.7 Otro ejemplo: un prog. Conversión binario a decimal.
  - 5.1.8 A vueltas con las cadenas, inversión de una.
  - 5.1.9 Subcadenas: el operador de corte
  - 5.1.10 Una aplicación: correo electrónico personalizado
  - 5.1.11 Referencias a cadenas

Escapes

- Las cadenas vistas hasta hoy, contenían caracteres con representación trivial.

- Suponga que quiera que una cadena incluya un salto de línea?

```
>>> a = 'una↵
  File "<string>", line 1
    'una
       ^
SyntaxError: invalid token
```

- Veremos como insertar caracteres especiales en las cadenas, y como escapar a las representaciones usuales de caracteres.

Secuencias de ESCAPE

- **Solución: «\» carácter de escape.**

- Indica que el próximo carácter NO tiene la representación trivial.

- Si por ejemplo le sigue la letra «n» la nueva interpretación es un salto de línea

```
>>> a = 'una\ncadena' ↲
>>> print a ↲
una
cadena
```

- Esta secuencia de escape ocupa el lugar de un carácter internamente, se codifica con un numero ascii.

Secuencias de ESCAPE

- Secuencias de escape para caracteres de control

  - \n salto de línea
  - \t tabulador horizontal
  - \000 carácter cuyo código ascii en octal es 000
  - \xhh carácter cuyo código ascii en hexad. es hh

  - Ejs:

```
>>> print 'uno\tdos\ttres'
uno     dos     tres
>>> print '1\t2\t3'
1       2       3
```

```
>>> print 'A\nB\012C\x0aD'
A
B
C
D
```

Secuencias de ESCAPE

- Ciertos caracteres NO TIENEN representación directa en una cadena, por ejemplo para expresar la «\» como haría?

```
>>> print 'a\\b'
a\b
```

- Y si quisiera encerrar algo entre comillas? en una cadena?

```
>>> print 'Munich\'72'
Munich'72
>>> print "Una \"cosa\" rara."
Una "cosa" rara.
```

Secuencias de ESCAPE

- Cuales serán las salidas de las siguientes secuencias:

```
>>> print '\\n'
```

```
>>>print '\157\143\164\141\154'
```

```
>>>print r'u\n'
```

- Las cadenas con una «r» previa a las comillas, **no interpretan caracteres de escape**. Son cadenas directas RAW STRINGS.
- También existen cadenas multilineas comienzan y terminan con TRIPLE COMILLAS. ‘ ‘ ‘ ….PRUEBALO!

- 5.1 Cadenas
  - 5.1.1 Lo que ya sabemos
  - 5.1.2 Escapes
  - **5.1.3 Longitud de una cadena**
  - 5.1.4 Indexación
  - 5.1.5 Recorrido de cadenas
  - 5.1.6 Un ejemplo: un contador de palabras
  - 5.1.7 Otro ejemplo: un prog. Conversión binario a decimal.
  - 5.1.8 A vueltas con las cadenas, inversión de una.
  - 5.1.9 Subcadenas: el operador de corte
  - 5.1.10 Una aplicación: correo electrónico personalizado
  - 5.1.11 Referencias a cadenas

Longitud de una CADENA

Ing. Ventre, Luis O.

- **La función LEN**. Abreviatura del inglés length.

- Esta función devuelve la longitud de una cadena, es decir el número de caracteres que la FORMAN.

- Ejemplos:

```
>>> len ('abc')
```
3

```
>>> len ('a\n')
```
2

```
>>> len ('')
```
0

```
>>> len (' ')
```
1

- 5.1 Cadenas
  - 5.1.1 Lo que ya sabemos
  - 5.1.2 Escapes
  - 5.1.3 Longitud de una cadena
  - **5.1.4 Indexación**
  - 5.1.5 Recorrido de cadenas
  - 5.1.6 Un ejemplo: un contador de palabras
  - 5.1.7 Otro ejemplo: un prog. Conversión binario a decimal.
  - 5.1.8 A vueltas con las cadenas, inversión de una.
  - 5.1.9 Subcadenas: el operador de corte
  - 5.1.10 Una aplicación: correo electrónico personalizado
  - 5.1.11 Referencias a cadenas

Indexación

- Podemos acceder a cada ELEMENTO de una cadena, a través de un operador de INDEXACION.

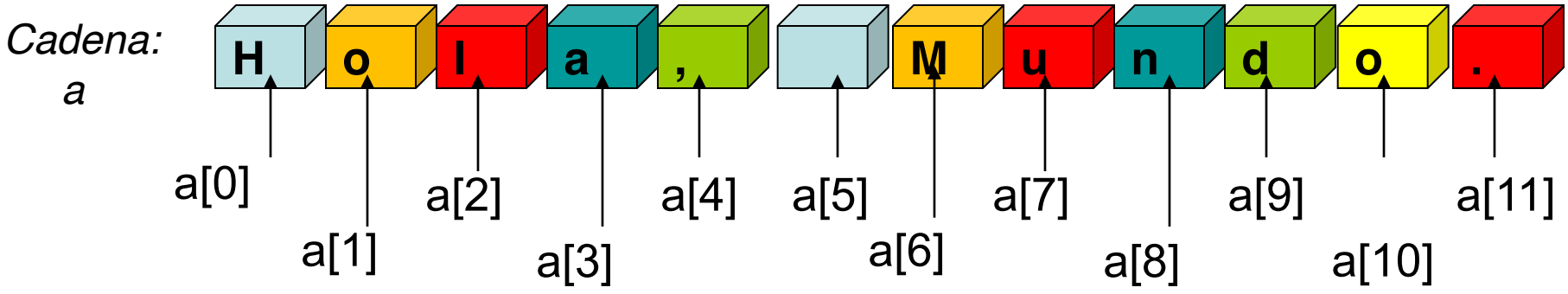
- El índice del elemento que se quiere acceder se debe encerrar entre **CORCHETES.**

```
a = 'Hola, Mundo.'
```

- **Si a es la cadena vista, a[i] será el elemento de la posición i+1. Ya que el primer elemento tiene subíndice 0.**

Indexación

- Veamos los índices de la siguiente cadena:




```
>>>'Hola, Mundo.'[0]
'H'
```

```
>>>'Hola, Mundo.'[1]
'o'
```

```
>>>a='Hola, Mundo.'
>>>a[2]
'l'
```

```
>>>a[len(a) -1]
'.'
```

Indexación

- Si indexamos mal, python se queja!

```
>>> a[len(a)] ↵
Traceback (innermost last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
```

- También puedes usar índices negativos, los cuales comienzan desde el final de la cadena, por lo cual:

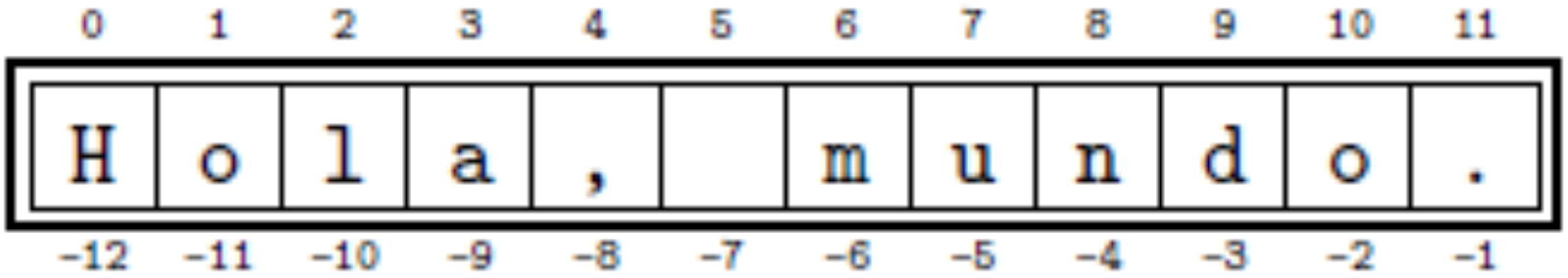
```
>>>'Hola, Mundo.'[-3]
```

Indexación

- Veamos un ejemplo

```
>>>'Hola, Mundo.'[-1]
'.'
```

- Podría decirse que existe una doble indexación…los índices finales serán:



- 5.1 Cadenas
  - 5.1.1 Lo que ya sabemos
  - 5.1.2 Escapes
  - 5.1.3 Longitud de una cadena
  - 5.1.4 Indexación
  - 5.1.5 Recorrido de cadenas
  - 5.1.6 Un ejemplo: un contador de palabras
  - 5.1.7 Otro ejemplo: un prog. Conversión binario a decimal.
  - 5.1.8 A vueltas con las cadenas, inversión de una.
  - 5.1.9 Subcadenas: el operador de corte
  - 5.1.10 Una aplicación: correo electrónico personalizado
  - 5.1.11 Referencias a cadenas

Recorrido de cadenas

- Una propiedad interesante de las cadenas es que pueden recorrerse de izquierda a derecha:

```
>>> for caracter in "mi cadena":
        print caracter
```

```
>>> for c in "la cadena":
...     print c
...
l
a

c
a
d
e
n
a
>>> for c in "la cadena":
...     print c,
...
l a   c a d e n a
>>>
```

- Ojo…comprender diferencia con recorrido por subindices!

Recorridos de cadenas

- Otra forma es utilizar subíndices:

```
>>> a= “mi cadena”
>>> for i in range(len(a))
        print a[i]
```

- Y si quiero imprimir la cadena al reves?

```
>>> a= “mi cadena”
>>> for i in range(len(a))
        print a[len(a)-i-1]
```

```
>>> a= “mi cadena”
>>> for i in range(len(a)-1,-1,-1)
        print a[i]
```

- 5.1 Cadenas
  - 5.1.1 Lo que ya sabemos
  - 5.1.2 Escapes
  - 5.1.3 Longitud de una cadena
  - 5.1.4 Indexación
  - 5.1.5 Recorrido de cadenas
  - 5.1.6 Un ejemplo: un contador de palabras
  - 5.1.7 Otro ejemplo: un programa para conversión binario a decimal.
  - 5.1.8 A vueltas con las cadenas, inversión de una.
  - 5.1.9 Subcadenas: el operador de corte
  - 5.1.10 Una aplicación: correo electrónico personalizado
  - 5.1.11 Referencias a cadenas

Programa Convertir Binario en Decimal

- El objetivo del programa es recibir una cadena de 0 y 1 y muestre el numero correspondiente al valor DECIMAL de la cadena interpretada como número binario.…
- A tener en cuenta, podemos recorrer la cadena de izquierda a derecha.
- Sumando cada aporte de cada bit, pero OJO, esta invertida la posición con el índice.
- La cadena ‘110’, el primer 1, esta en la posición 0, pero aporta 1 por 2 elevado a la (3-1).

Programa Convertir Binario en Decimal

- Podemos guardar en una variable e ir cambiando el valor de n.

decimal.py

```
1  bits = raw_input('Dame un número binario: ')
2
3  n = len(bits)
4  valor = 0
5  for bit in bits:
6      if bit == '1':
7          valor = valor + 2 ** (n-1)
8      n -= 1
9
10 print 'Su valor decimal es', valor
```

- 5.1 Cadenas
  - 5.1.1 Lo que ya sabemos
  - 5.1.2 Escapes
  - 5.1.3 Longitud de una cadena
  - 5.1.4 Indexación
  - 5.1.5 Recorrido de cadenas
  - 5.1.6 Un ejemplo: un contador de palabras
  - 5.1.7 Otro ejemplo: un programa para conversión binario a decimal.
  - 5.1.8 A vueltas con las cadenas, inversión de una cadena.
  - 5.1.9 Subcadenas: el operador de corte
  - 5.1.10 Una aplicación: correo electrónico personalizado
  - 5.1.11 Referencias a cadenas

Inversión de una cadena

- Teniendo en mente, el uso del operador «+» CONCATENACION en cadenas.
- El programa que nos planteamos resolver, tiene por objetivo luego de ingresar una cadena, imprimir en pantalla la cadena inversa, al revés de la ingresada.
- Para ello, nos valdremos de una cadena auxiliar, inicialmente vacía.
- A esta cadena vacía le iremos agregando los caracteres leídos de la cadena ingresada.

Inversión de una cadena

- Podemos ver el resultado:

inversion.py

```
1 cadena = raw_input('Introduce una cadena: ')
2
3 inversion = ''
4 for caracter in cadena:
5     inversion = caracter + inversion
6
7 print 'Su inversión es:', inversion
```

- 5.1 Cadenas
  - 5.1.1 Lo que ya sabemos
  - 5.1.2 Escapes
  - 5.1.3 Longitud de una cadena
  - 5.1.4 Indexación
  - 5.1.5 Recorrido de cadenas
  - 5.1.6 Un ejemplo: un contador de palabras
  - 5.1.7 Otro ejemplo: un programa para conversión binario a decimal.
  - 5.1.8 A vueltas con las cadenas, inversión de una cadena.
  - 5.1.9 Sub-cadenas: el operador de corte
  - 5.1.10 Una aplicación: correo electrónico personalizado
  - 5.1.11 Referencias a cadenas

- Nos proponemos desarrollar un programa que dada una cadena y dos índices i y j, el mismo muestre la sub cadena formada desde el índice i hasta el j sin incluir este último.

- La idea seria construir una nueva cadena inicialmente vacía.

- Y agregarle los elementos desde el i, hasta el j – 1.

- Veamos una primera aproximación:

subcadena_3.py

subcadena.py

```
1 cadena = raw_input('Dame una cadena: ')
2 i = int(raw_input('Dame un número: '))
3 j = int(raw_input('Dame otro número: '))
4
5 subcadena = ''
6 for k in range(i, j):
7     subcadena += cadena[k]
8
9 print 'La subcadena entre %d y %d es %s.' % (i, j, subcadena)
```

- Pero que pasa si pongo cualquier subindice en i y j?

- Corrigiendo eso….

subcadena_4.py

subcadena.py

```
1  cadena = raw_input('Dame una cadena: ')
2  i = int(raw_input('Dame un número: '))
3  j = int(raw_input('Dame otro número: '))
4
5  if j > len(cadena):
6    final = len(cadena)
7  else:
8    final = j
9  subcadena = ''
10 for k in range(i, final):
11    subcadena += cadena[k]
12
13 print 'La subcadena entre %d y %d es %s.' % (i, j, subcadena)
```

- Esta operación es muy utilizada en las cadenas…por lo que python nos ofrece un operador predefinido llamado **OPERADOR DE CORTE**.

- El mismo tiene una notación particular. El mismo se denota con « **:** »

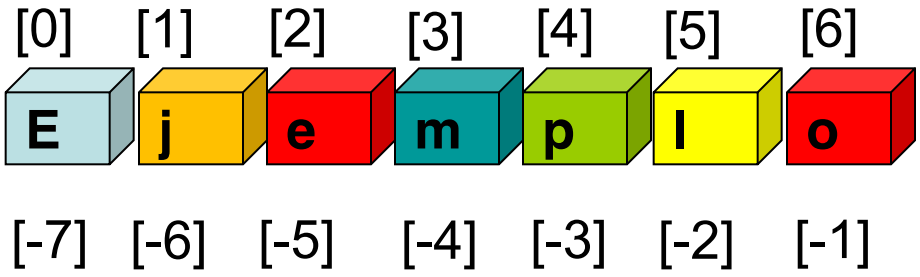
```
>>> a = “Ejemplo”
>>> a [ i : j ]
```

- La expresión **a[i : j]** indica la sub-cadena formada por los caracteres que comienzan en el índice i hasta j - 1

- Como existían, dos indexaciones las siguientes subcadenas son idénticas: …puede observarlo?

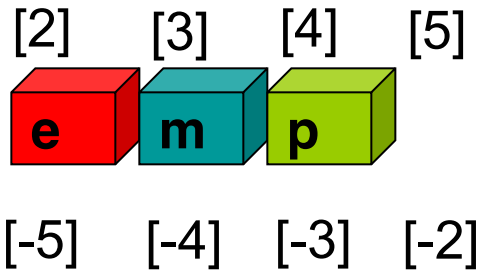
```
>>> a = “Ejemplo”

>>> a [ 2 :  5]
>>> a [-5 :  5]
>>> a [ 2 : -2]
>>> a [-5 : -2]
```



- Recuerda el ultimo elemento no se incluye!...

```
>>> a [ 2 :  5]
>>> a [-5 :  5]
>>> a [ 2 : -2]
>>> a [-5 : -2]
```



- Además si omites el primer índice de corte, se inicia por **DEFECTO en 0**.

```
>>> a [  :  j ]
```

```
>>> a [ 0 : j ]
```

- Y si se omite el ultimo índice, el final será el **final de la cadena**

```
>>> a [ i :  ]
```

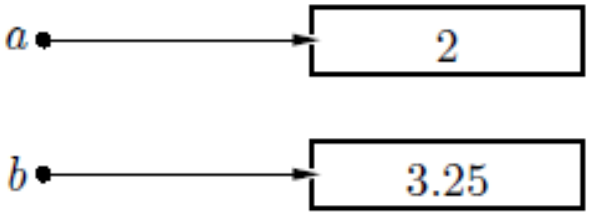
```
>>> a [ i :  len(a)]
```

- 5.1 Cadenas
  - 5.1.1 Lo que ya sabemos
  - 5.1.2 Escapes
  - 5.1.3 Longitud de una cadena
  - 5.1.4 Indexación
  - 5.1.5 Recorrido de cadenas
  - 5.1.6 Un ejemplo: un contador de palabras
  - 5.1.7 Otro ejemplo: un programa para conversión binario a decimal.
  - 5.1.8 A vueltas con las cadenas, inversión de una.
  - 5.1.9 Subcadenas: el operador de corte
  - 5.1.10 Una aplicación: correo electrónico personalizado
  - 5.1.11 Referencias a cadenas

Referencias a cadenas

- Cuando comenzamos a analizar variables, lo hicimos con **diagramas de cajas**:

- Si asignabamos a una variable a y b los siguientes valores implicaba:



- Las flechas se llaman punteros o referencias.

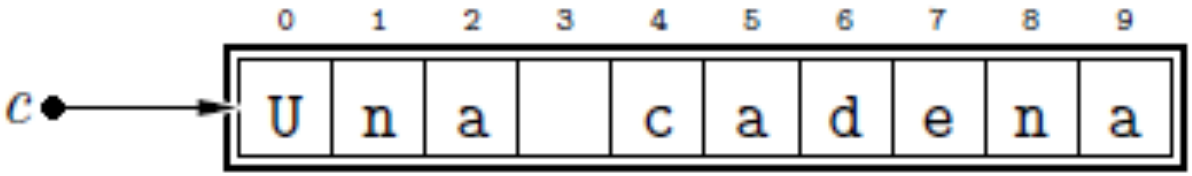
- A apunta al valor de 2 y b al de 3.25

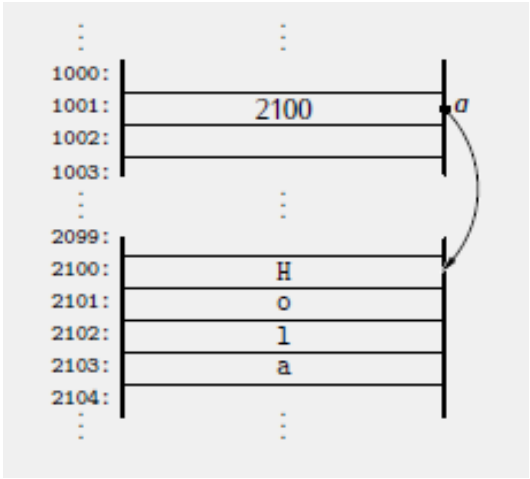
Referencias a cadenas

- Ahora si hacemos la siguiente asignación que pasara?

```
>>> c = “una cadena”
>>> a = “Hola”
```

- Se dice que la variable c apunta a una cadena:





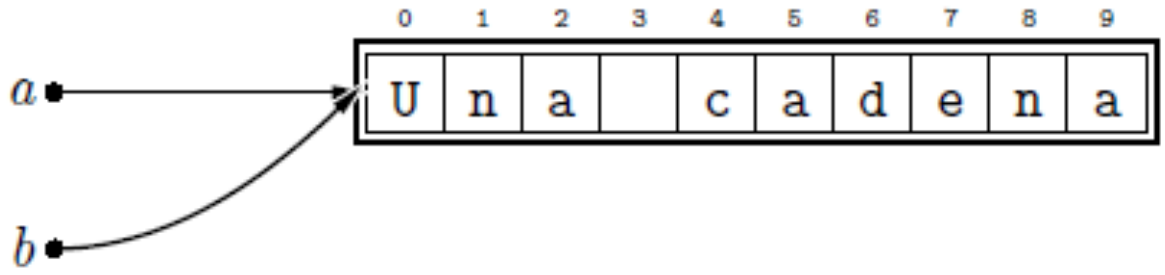
- Que las variables tengan referencias a los datos y NO los datos mismos, es muy util, para aprovechar la MEMORIA!

Referencias a cadenas

- Supongamos que ahora hacemos:

```
>>> a = b
```

- Tanto **a** como **c** apuntan a la misma cadena!:



- Solo se copio su referencia y no todo el contenido!! Esto es independiente de la longitud de la cadena!, y casi instantaneo.

Referencias a cadenas

- Por lo tanto una asignación solo ALTERA EL VALOR DE UN PUNTERO.

- Mientras que operaciones como CONCATENACION, y el OPERADOR DE CORTE, reservan ESPACIO DE MEMORIAS NUEVOS!

```
>>> a = 'otra'
>>> b = 'cadena'
>>> c = a + b
```