IWI 131- PROGRAMACIÓN

Ayudantía 6

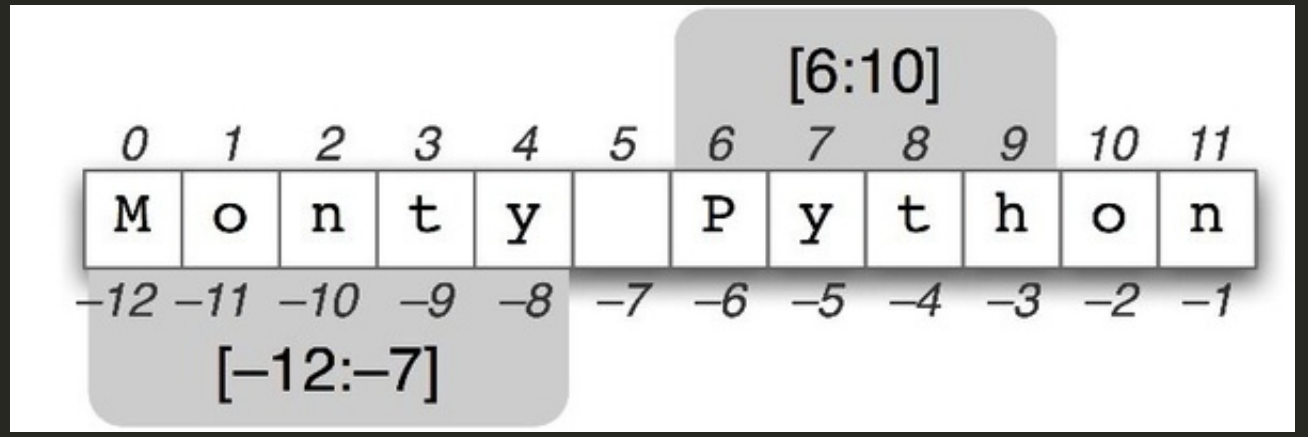
Ayudante: Anastasiia Fedorova

Paralelo: 212

Fecha: 13.05.2020

STRINGS

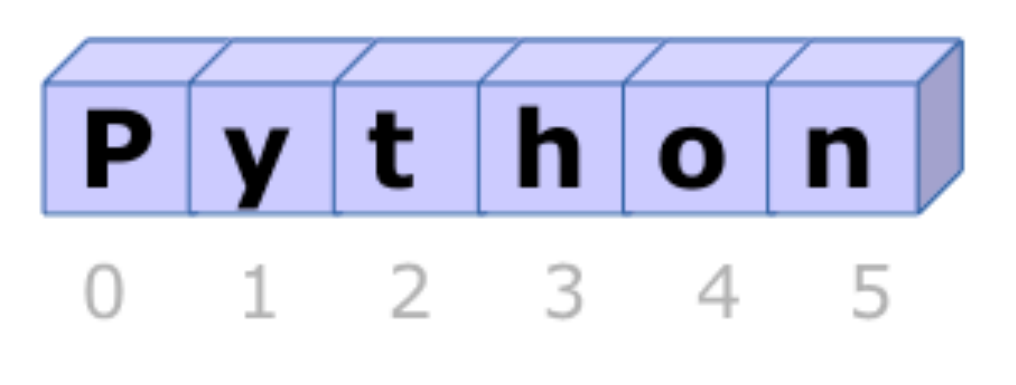
Los strings, son, esencialmente, listas de caracteres, unidos uno a uno. Por lo que, al ejecutar operaciones sobre estos, tenemos que pensar en los elementos individuales, unidos como una cadena. Para aceder a *un* elemeto de cadena, se usa `s[indice]`, siendo `s` – el string y `indice` – un entero de posición en este.




```
"Monty Python"
```

INDEXACIÓN: DE 0 A (N-1)

En Python, si contamos de *izquierda a derecha*, los elementos empiezan desde 0 – es la forma mas común de indexar los elementos en listas, que persiste en muchos lenguajes.



¿Qué imprime esto?

```
s = "Python"
N = len(s)
print(s[0])
print(s[N - 1])
print(s[N])
```

INDEXACIÓN: DE 0 A (N-1)

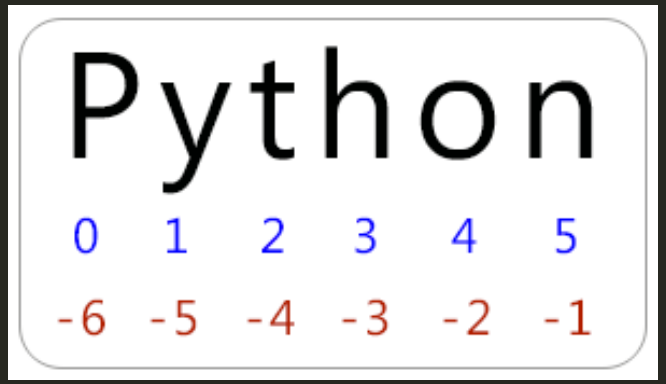
Como partimos de 0, no podremos llegar a indice N – sino a (N-1). Al acceder fuera del rango - en posición N - su programa va a crashear con el mensaje “Index Error: index out of range”

```
s = "Python"
N = len(s)
print(s[0])
print(s[N - 1])
print(s[N])
```

```
>>> s = "Python"
>>> N = len(s)
>>> print(N)
6
>>> print(s[0])
P
>>> print(s[N-1])
n
>>> print(s[N])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

INDEXACIÓN: DE -1 A -N

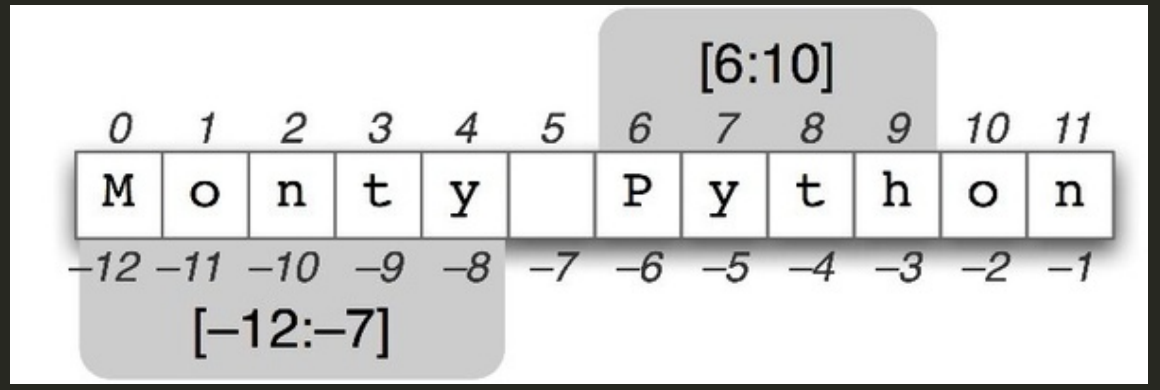
Dado que Python es un lenguaje muy flexible, tambien presenta una notación muy interesante para la indexación – se toma la última posición del string como -1, por lo que podemos referirnos a cualquier posición en el string de manera relativa a su final. Así, el inicio de string tendrá valor posicional de –N.



El ejemplo anterior, con 2 notaciones

SUBSTRINGS

Adicionalmente, podemos acceder a unas partes de string, llamadas “slice” – es un corte de string, una subcadena. Su sintaxis es `s[inicio:final]`, siendo `inicio` y `final` los índices de string. No se incluirá en el substring el carácter, que está en la posición final. Se puede hacer slice solo de izquierda a la derecha [-12:-7] y no [-7:-12]



¿Cuáles son todas las maneras de obtener Monty? ¿Python?

SLICES

Existen *varios* tipos de slices. Algunos de estos son:

- `s[i:f]` – lo más común. Entrega substring que inicia en `i` y termina en posición `f`.

```
s = "Ayudantia 6"
print(s[0:5])
```

```
>>> print(s[0:5])
Ayuda
```

- `s[i:]` – entrega substring que inicia en `i` termina donde termina el string original.
- `s[:f]` - – entrega substring que de incio de string original, hasta posición `f`.

```
s = "Hurones son muy lindos"
print(s[:7])
print(s[7:])
```

```
>>> print(s[:7])
Hurones
>>> print(s[7:])
 son muy lindos
>>>
```

SLICES

Existen *varios* tipos de slices. Algunos de estos son:

- `s[:]`

```
s = "Hurones son muy lindos"
print(s[:])
```

```
>>> print(s[:])
Hurones son muy lindos
```

- `s[i:f:paso]` – imprime substring que inicia en `i` y termina de `f`, pero con cierto paso

```
     X123456789123456789123
s = "Hurones son muy lindos"
print(s[0:len(s):1])
print(s[0:len(s):9])
```

```
>>> print(s[0:len(s):1])
Hurones son muy lindos
>>> print(s[0:len(s):9])
Hon
```

* y +

Para los strings tenemos dos operaciones especiales: concatenación y repetición

```
str_1 = 'Hola '
str_2 = 'mundo'
str_1 + str_2 # El resultado es: Hola mundo
```

```
>>> print('rojo' + 'amarillo')
rojoamarillo
>>> print ('rojo' * 3)
rojorojorojo
>>> print 'rojo' + 3
Traceback (most recent call last):
File "", line 1, in
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

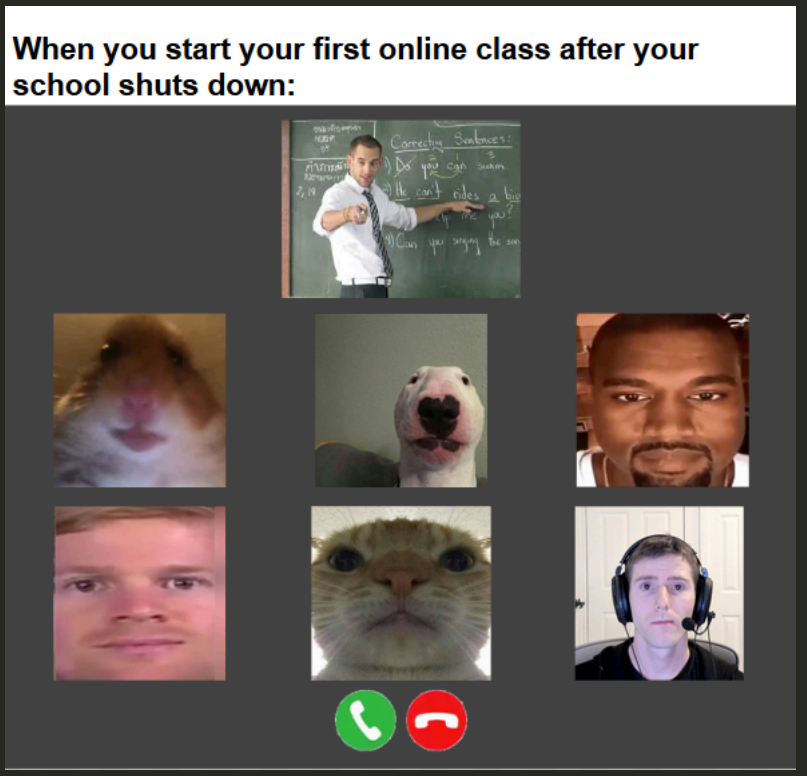
La concatenación fusiona dos strings (o más) en uno solo.

El operador * crea un string con n repeticiones de string original.

EJERCICIO 1

Para entrar al Zoom, muchos de los profesores filtran los correos, con que los alumnos entran a la reunión. Se permiten solo los correos, que contienen el sufijo de @sansano.usm.cl.

Crear un programa, que revisa que el correo ingresado sea el permitido. Además, crea un nickname para el alumn@ – este corresponde a su direccion de correo, sin el hosting, en caso de que sea correcto su mail. Utilice solamente slices – no se permite usar la función `in()` ni `split()`.



Input:

anastasiia.fedorova@sansano.usm.cl

Output:

Correcto

anastasiia.fedorova

EJERCICIO 1: SOLUCIÓN

```
correo = input("Ingrese el correo: ")
#@sansano.usm.cl  va de -15 hasta fin de
string (en caso de correo correcto)
hosting  = correo[-15:]
if(hosting == "@sansano.usm.cl"):
    nickname = correo[:-15]
    print("Correo valido")
    print("Su nickname es: " + nickname)
else:
    print("Correo invalido.")
```

LEN(),IN

La función `len(s)`, proviene de *length*, retorna un entero, que corresponde al largo del string s.

```
s = input("Ingrese su nombre")
if len(s):
    print("Hola " + s)
else:
    print("No se ingreso nada.")
```

El operador `in` retorna `True` si se encuentra el substring en un string original y `False` en otro caso.

```
kiosko_de_U = ”Kiosko tiene Super8 y Prestigio"
if ”Prestigio" in kiosko_de_U:
    print(”Hoy tomo te con Prestigio")
```

CICLO FOR

El ciclo `for`, a diferencia de ciclo `while`, no utiliza la condición logica para determinar cuántas veces ejecutar el bloque de instrucciones. En vez de ello, `for` corre un número predeterminado de veces. Este numero corresponde a la cantidad de elementos en la variable recorrible – en general, una lista.

```
s = "Universidad”
count = 0
for i in s:
    count += 1
print("El string tiene", count, "letras")
```

Simulación de función len()

```
veces = int(input("¿Cuántas veces? "))
veces = "1" * veces
for i in veces:
    print("Hola")
```

Este codigo usa la repetición de strings, para evitar el uso de listas, generadas por `range`. Se puede quitar la 2da linea y en for correr `in range(veces)`. El resultado es el mismo – el loop corre “veces” veces.

EJERCICIO 2

C1 2018-1.

La exposición interactiva "Bytes" de un museo de la Historia de la Computación, simula cómo se interpretan los Bytes de un computador. Para ello, la instalación permite que los visitantes presionen 8 grandes interruptores para definir el estado de 8 bits en "0" (apagado) ó "1" (encendido). **1 Byte equivale a 8 bits**. Luego, el software transforma los 8 bits del Byte en un número entero y lo presenta en las pantallas gigantes del museo.

Notar que cada bit del Byte representa una potencia de 2 (del 7 al 0) que se activa/desactiva dependiendo de si está en el valor 1 o 0 respectivamente y que se van sumando hasta alcanzar el número en formato decimal. Para esto, el bit de más a la derecha se multiplica por 2^0, el contiguo por 2^1 y así hasta el de más a la izquierda que se multiplica por 2^7. Por ejemplo si se desea calcular en decimal el número 10000101 esto se obtiene mediante:

1	0	0	0	0	1	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$= 1 \cdot 2^7 + 1 \cdot 2^2 + 1 \cdot 2^0 = 128 + 4 + 1 = 133$$

Así otro ejemplo sería: $00000011 = 1 \cdot 2^1 + 1 \cdot 2^0 = 3$.

Crear una función, que recibe un string de un número binario y que retorna un entero, correspondiente a este binario, pero en base 10. Asumen que se ingresa la cantidad correcta de bits.

EJERCICIO 2: SOLUCIÓN

```python
def b10(bin):
    decimal = 0
    #invierte el string
    bin = bin[::-1]
    for i in range(0, len(bin)):
        decimal += (2 ** i) * int(bin[i])
    return decimal

num = input("Ingrese el binario: ")
print("Su binario en base 10 es:", b10(num))
```

Usando range y slice

```python
def b10(bin):
    decimal = 0
    n = len(bin) - 1
    for i in bin:
        decimal += (2 ** n) * int(i)
        n -= 1
    return decimal

num = input("Ingrese el binario: ")
print("Su binario en base 10 es:", b10(num))
```

Usando recorrido directo de string

EJERCICIO 3

progra.usm.cl

Dado un mensaje, se debe calcular su costo para enviarlo por telégrafo. Para esto se sabe que cada letra cuesta $10, los caracteres especiales que no sean letras cuestan $30 y los dígitos tienen un valor de $20 cada uno. Los espacios no tienen valor.

Su mensaje debe ser un string, y las letras del castellano (ñ, á, é, í, ó, ú) se consideran caracteres especiales.

```
Mensaje: Feliz Aniversario!
Su mensaje cuesta $190
```

EJERCICIO 3: SOLUCIÓN

```
def telegrafo(s):
    costo = 0
    for letra in s:
        if letra in "0123456789":
            costo += 20
        #letras normales - minusculas van de 97 a 122 en ASCII
        elif  97 <= ord(letra.lower()) <= 122:
            costo += 10
        else:
            if letra != " ":
    return costo#especiales
                costo += 30


msg = input("Ingrese su mensaje: ")
print("El costo de su menasje es: $" + str(telegrafo(msg)))
```

EJERCICIO 4: LA BUENA JUNA

Muchos de los estudiantes de la USM usan su tarjeta JUNAEB para pagar en el casino por su almuerzo. Un día a un estudiante se le olvidó su clave, por lo que tenía dos opciones - o intentar recordar la clave o pasar hambre.

La máquina que acepta el pago funciona de la siguiente manera – si se ingresa un dígito correcto, lo guarda y si el dígito fue incorrecto, lo ignora y repite el proceso. Una vez que se logra ingresar los 4 dígitos correctos, el pago pasa al sistema.

Emular el funcionamiento de la máquina, si esta imprime * por cada dígito que todavía no se ingresa correctamente, y la contraseña se ingresa de izquierda a derecha (1234 es 1,2,3,4; no al revés)

Input:	Output:
0	***_
1	**1_
2	*12_
3	123_
9	123_
4	1234

EJERCICIO 4: SOLUCIÓN

```
from random import randint
def generar_password():
    password = ''
    for i in range(0, 4):
        password += str(randint(0, 9))
    return password

correcto = True
password = generar_password()
digitos_correctos = ''
i = 0
print("La contraseña que se debe ingresar es " + password)
while correcto:
    #end='' quita el salto de linea de print
    print("*" * (3 - len(digitos_correctos)) + digitos_correctos, end='')
    digito = input()
    if digito == password[i]:
        i += 1
        digitos_correctos +=  digito
    if len(digitos_correctos) == 4:
        correcto = False
print("Su pago se efectuo exitosamente.")
```