**PYTHON**

**Cambiar un número de entero a no entero:**

*variable = float(variableentera)*

***Convertir de no entero a entero***

*variable = int(variablefloat)*

***Floored division: hace una división y te devuelve un núm entero si los dos son enteros y sino n float:***

*5//2*

OJO CON LA TOLERANCIA. A veces le preguntas si x==2.5 pero en realidad es 2.50000, o una movida así con el tema de los floats y pedirle enteros

**Hacer una lista:**

*nombrelista = [ ]*

(si le pon[es parentesis es un tupple, es inmutable)

**Llamar a un elemento de la lista (ojo que empieza desde la posición cero)**

*print(nombrelista[0])*

Para pedir el último valor ponés -1

**Llamar un subgrupo de la lista:**

*print(nombrelista[2:4])*

Ojo que el último (posición 4) no lo agarra

**Agregar un elemento a la lista:**

*nombrelista.append(29)*

**Un loop: agarrar elemento por elemento y mostrarlo:**

*for x in mylist:*

*print(x)*

**Un loop: agregar elementos a la lista, acá agrega el 4 y el 5**

*a = [1,2,3]*

*a += [4,5]*

(o sin corchetes)

**Quedarte con los strings de una lista:**

*# Imaginemos que tenemos una lista de nombres no ordenados que de alguna manera se incluyeron algunos números al azar.*

*# Para este ejercicio, queremos imprimir la lista alfabética de nombres sin los números.*

*# Esta no es la mejor manera de hacer el ejercicio, pero ilustrará un montón de técnicas.*

*names = ["John", 3234, 2342, 3323, "Eric", 234, "Jessica", 734978234, "Lois", 2384]*

*print("Number of names in list: {}".format(len(names)))*

*# Primero eliminamos esos números*

*new\_names = []*

*for n in names:*

*if isinstance(n, str):*

*# Si n es string, agregar a la lista*

*# Notar la doble sangría*

*new\_names.append(n)*

**Eliminar un (dos) elemento de la lista:**

*lista[0:2]=[ ]*

**OPERACIONES BÁSICAS:**

**La potencia es con** *\*\**

**Sumar y asignar:** *+=* ***.* Lo que hace es sumarle tres a la variable y sobreescribirla con el resultado.**

**Definir una funcion:**

*def say\_hello()*

*print(‘Hello, World’)*

ejemplo:

*def calificar(sujeto, adjetivo):*

*print("{} es {}".format(sujeto, adjetivo))*

*calificar ("Juan", "capo")*

**Testear si funciona la función:**

*print(callable(say\_hello)* y tendría que salir true

return

es como una función que devuelve un valor. Este valor a menudo no es visto por el usuario humano, pero puede ser usado por la computadora en otras funciones.

Ej:

def add\_three(num):

    return num + 3

**CURSO UNSAM**

Usá el guión bajo (underscore, \_) para referirte al resultado del último cálculo.

**Ejecutar** en una terminal de Windows:

C:\SomeFolder>hello.py

hello world

C:\SomeFolder>c:\python36\python hello.py

hello world

A veces es conveniente especificar un bloque de código que no haga nada. El comando pass se usa para eso.

if a > b:

pass

else:

print('No ganó a')

x + y Suma

x - y Resta

x \* y Multiplicación

x / y División (da un float, no un int)

x // y División entera (da un int)

x % y Módulo (resto)

x \*\* y Potencia

abs(x) Valor absoluto

x << n Desplazamiento de los bits a la izquierda

x >> n Desplazamiento de los bits a la derecha

x & y AND bit a bit.

x | y OR bit a bit.

x ^ y XOR bit a bit.

~x NOT bit a bit.

import math

a = math.sqrt(x)

b = math.sin(x)

c = math.cos(x)

d = math.tan(x)

e = math.log(x)

x < y Menor que

x <= y Menor o igual que

x > y Mayor que

x >= y Mayor o igual que

x == y Igual a

x != y No igual a

Con esto en mente, ¿podrías explicar el siguiente comportamiento?

>>> bool("False")

True

>>>

Normalmente las cadenas de caracteres solo ocupan una linea. Las comillas triples nos permiten capturar todo el texto encerrado a lo largo de múltiples lineas:

# Comillas triples

c = '''

Yo no tengo en el amor

Quien me venga con querellas;

Como esas aves tan bellas

Que saltan de rama en rama

Yo hago en el trébol mi cama

Y me cubren las estrellas.

'''

**Código de escape**

Los códigos de escape (escape codes) son expresiones que comienzan con una barra invertida, \ y se usan para representar caracteres que no pueden ser fácilmente tipeados directamente con el teclado. Estos son algunos códigos de escape usuales:

'\n' Avanzar una línea

'\r' Retorno de carro El retorno de carro (código '\r') mueve el cursor al comienzo de la línea pero sin avanzar una línea. El origen de su nombre está relacionado con las máquinas de escribir.

'\t' Tabulador

'\'' Comilla literal

'\"' Comilla doble literal

'\\' Barra invertida literal

### Indexación de cadenas

Las cadenas funcionan como los vectores multidimensionales en matemática, permitiendo el acceso a los caracteres individuales. El índice comienza a contar en cero. Los índices negativos se usan para especificar una posición respecto al final de la cadena.

a = 'Hello world'

b = a[0] # 'H'

c = a[4] # 'o'

d = a[-1] # 'd' (fin de cadena)

También se puede rebanar (slice) o seleccionar subcadenas especificando un range de índices con :.

d = a[:5] # 'Hello'

e = a[6:] # 'world'

f = a[3:8] # 'lo wo'

g = a[-5:] # 'world'

### Operaciones con cadenas

Concatenación, longitud, pertenencia y replicación.

# Concatenación (+)

a = 'Hello' + 'World' # 'HelloWorld'

b = 'Say ' + a # 'Say HelloWorld'

# Longitud (len)

s = 'Hello'

len(s) # 5

# Test de pertenencia (in, not in)

t = 'e' in s # True

f = 'x' in s # False

g = 'hi' not in s # True

# Replicación (s \* n)

rep = s \* 5 # 'HelloHelloHelloHelloHello'

### Métodos de las cadenas

Las cadenas en Python tienen métodos que realizan diversas operaciones con este tipo de datos.

Ejemplo: sacar (strip) los espacios en blanco sobrantes al inicio o al final de una cadena.

s = ' Hello '

t = s.strip() # 'Hello'

Ejemplo: Conversión entre mayúsculas y minúsculas.

s = 'Hello'

l = s.lower() # 'hello'

u = s.upper() # 'HELLO'

Ejemplo: Reemplazo de texto.

s = 'Hello world'

t = s.replace('Hello' , 'Hallo') # 'Hallo world'

**Más métodos de cadenas:**

Los strings (cadenas) ofrecen una amplia variedad de métodos para testear y manipular textos. Estos son algunos de los métodos:

s.endswith(suffix) # Verifica si termina con el sufijo

s.find(t) # Primera aparición de t en s (o -1 si no está)

s.index(t) # Primera aparición de t en s (error si no está)

s.isalpha() # Verifica si los caracteres son alfabéticos

s.isdigit() # Verifica si los caracteres son numéricos

s.islower() # Verifica si los caracteres son minúsculas

s.isupper() # Verifica si los caracteres son mayúsculas

s.join(slist) # Une una lista de cadenas usando s como delimitador

s.lower() # Convertir a minúsculas

s.replace(old,new) # Reemplaza texto

s.split([delim]) # Parte la cadena en subcadenas

s.startswith(prefix) # Verifica si comienza con un sufijo

s.strip() # Elimina espacios en blanco al inicio o al final

s.upper() # Convierte a mayúsculas

Los strings son "inmutables" o de sólo lectura. Una vez creados, su valor no puede ser cambiado. Esto implica que las operaciones y métodos que manipulan cadenas deben crear nuevas cadenas para almacenar su resultado.

### Ejercicio 1.16: Testeo de pertenencia (test de subcadena)[¶](https://programacionpython.ecyt.unsam.edu.ar/material/01_Introduccion/04_Strings/#ejercicio-116-testeo-de-pertenencia-test-de-subcadena)

Experimentá con el operador in para buscar subcadenas. En el intérprete interactivo probá estas operaciones:

>>> 'Naranja' in frutas

?

>>> 'nana' in frutas

True

>>> 'Lima' in frutas

?

>>>

### Ejercicio 1.21: Expresiones regulares

Una limitación de las operaciones básicas de cadenas es que no ofrecen ningún tipo de transformación usando patrones más sofisticados. Para eso vas a tener que usar el módulo re de Python y aprender a usar expresiones regulares. El manejo de estas expresiones es un tema en sí mismo. A continuación presentamos un corto ejemplo:

>>> texto = 'Hoy es 6/8/2020. Mañana será 7/8/2020.'

>>> # Encontrar las apariciones de una fecha en el texto

>>> import re

>>> re.findall(r'\d+/\d+/\d+', texto)

['6/8/2020', '7/8/2020']

>>> # Reemplazá esas apariciones, cambiando el formato

>>> re.sub(r'(\d+)/(\d+)/(\d+)', r'\3-\2-\1', texto)

'Hoy es 2020-8-6. Mañana será 2020-8-7.'

>>>

Para más información sobre el módulo re, mirá la [documentación oficial en inglés](https://docs.python.org/3/library/re.html) o algún [tutorial en castellano](https://rico-schmidt.name/pymotw-3/re/index.html). Es un tema que escapa al contenido del curso pero te recomendamos que mires en detalle en algún momento. Aunque no justo ahora. Sigamos...

### Comentario

A medida que empezás a usar Python es usual que quieras saber qué otras operaciones admiten los objetos con los que estás trabajando. Por ejemplo. ¿cómo podés averiguar qué operaciones se pueden hacer con una cadena?

Dependiendo de tu entorno de Python, podrás ver una lista de métodos disponibles apretando la tecla tab. Por ejemplo, intentá esto:

>>> s = 'hello world'

>>> s.<tecla tab>

>>>

Si al presionar tab no pasa nada, podés volver al viejo uso de la función dir(). Por ejemplo:

>>> s = 'hello'

>>> dir(s)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', ..., 'find', 'format',

'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',

'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition',

'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',

'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase',

'title', 'translate', 'upper', 'zfill']

>>>

dir() produce una lista con todas las operaciones que pueden aparecer luego del parámetro que le pasaste, en este caso s. También podés usar el comando help() para obtener más información sobre una operación específica:

>>> help(s.upper)

Help on built-in function upper:

upper(...)

S.upper() -> string

Return a copy of the string S converted to uppercase.

>>>

Los elementos de una cadena pueden ser separados en una lista usando el método split():

>>> line = 'Pera,100,490.10'

>>> row = line.split(',') #la coma indica el elemento que separa

>>> row

['Pera', '100', '490.10']

>>>

Para encontrar rápidamente la posición de un elemento en una lista, usá index().

nombres = ['Rosita','Manuel','Luciana']

nombres.index('Luciana') # 2

Si el elemento está presente en más de una posición, index() te va a devolver el índice de la primera aparición. Si el elemento no está en la lista se va a generar una excepción de tipo ValueError.

### Ordenar una lista

Las listas pueden ser ordenadas "in-place", es decir, sin usar nuevas variables.

s = [10, 1, 7, 3]

s.sort() # [1, 3, 7, 10]

# Orden inverso

s = [10, 1, 7, 3]

s.sort(reverse=True) # [10, 7, 3, 1]

# Funciona con cualquier tipo de datos que tengan orden

s = ['foo', 'bar', 'spam']

s.sort() # ['bar', 'foo', 'spam']

Usá sorted() si querés generar una nueva lista ordenada en lugar de ordenar la misma:

t = sorted(s) # s queda igual, t guarda los valores ordenados

odés acceder a los elementos de las listas anidadas usando múltiples operaciones de acceso por índice.

>>> items[0]

'spam'

>>> items[0][0]

's'

>>> items[1]

['Banana', 'Mango', 'Frambuesa', 'Pera', 'Granada', 'Manzana', 'Lima']

>>> items[1][1]

'Mango'

>>> items[1][1][2]

'n'

>>> items[2]

[101, 102, 103]

>>> items[2][1]

102

>>>

MANERA DE VER LO QUE ESTÁS ITERANDO (con un ejemplo de la clase):

for i,c in enumerate(cadena):

capadepenapa=capadepenapa+c

if c in ("aeiou"):

capadepenapa=capadepenapa+"p"+c #es lo mismo que poner capadepenapa += "p"+c

print(i,c,capadepenapa)

print(capadepenapa)

PARA HACER UN BLOQUE/SECCIÓN:

#%% Sección 1

Cómo chequear la versión de Python:

import sys

print(sys.version)

Cosas útiles para hacer loops:

cadena=”Hola”

for c in enumerate(cadena):

Acá el enumerate genera tuplas/pares de números y elementos del objeto. O sea genera (1, c), (2, a), etc.

l = [14,3,5,6]

for i in zip(cadena,l):

genera pares de los dos objetos, o sea, H 14, o 3, etc.

(Hace los pares mientras puede, cuando no tiene más pares frena.)

enumerate (cadena) es igual a poner zip(range(len(cadena)),cadena)