PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

INTRODUCCIÓN A LA PROGRAMACIÓN

Objetivos

- Materiales clase pasada
- Clase expositiva
  - Videos serán subidos después
- Introducir **listas** y **tuplas**
- Énfasis en listas, pues tuplas son versión reducida
- Demostrar con código

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

¿Qué es una lista?

- **¿Qué es?** Una **lista** (tipo list) es una secuencia finita y ordenada de valores (elementos)
  - Muy parecidas a las tuplas
  - Pero admiten cambio de contenido

- Ejemplo:

```
[10, 20, 30, 40, 50]
```

```
>>> # definamos algunas listas

>>> [10,20,30,40]
[10, 20, 30, 40]

>>> ["hola","mundo"]
['hola', 'mundo']

>>> [True,True,True]
[True, True, True]
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Pueden contener múltiples tipos

- Las listas pueden contener **int**s, **float**s, **string**s, **tuplas**, etc, incluso otras listas

- No hay restricción sobre los tipos de valores que pueden contener

- Ejemplo:

```
[5, 5.5, (6, 6.5), False]
```

```
>>> # definamos algunas listas

>>> x = [10,0,"hola",False,
-5.5]
>>> print(x)
[10, 0, 'hola', False, -5.5]

>>> x = [-1,True,(5,5,5)]
>>> print(x)
[-1, True, (5, 5, 5)]
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Listas de cero y un elementos

- **Lista vacía**: podemos definir una lista vacía de dos maneras

```
x = []
y = list()
```

Aquí **x** e **y** son listas vacías (son iguales para Python)

```
>>> a = []
>>> b = list()

>>> print(a,b)
[] []

>>> a == b
True
```

- **Lista de un elemento**: se coloca el elemento entre corchetes

```
x = [ 1 ]
```

```
>>> c = [1]
>>> print(c)
[1]
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Tuplas de cero y un elementos

- **Tupla vacía**: podemos definir una tupla vacía de dos maneras

```
x = ()
y = tuple()
```

Aquí **x** e **y** son tuplas vacías (son iguales para Python)

```
>>> a = ()
>>> b = tuple()

>>> print(a,b)
() ()

>>> a == b
True
```

- **Tupla de un elemento**: se coloca el elemento entre paréntesis, **pero con coma**

```
x = ( 1, )
```

```
>>> c = (1,)
>>> print(c)
(1,)
```

Concatenar listas

- La **concatenación** entre **listas** y entre **tuplas** se hace igual que con **strings**

- Usamos el operador **+** para concatenar

```
[1,2,3] + [4,5,6]
(1,2,3) + (4,5,6)
```

- *No podemos concatenar listas a strings, tuplas u otros tipos de datos*

```
>>> [1,2,3] + [4,5,6]
[1, 2, 3, 4, 5, 6]

>>> [] + [] + [] + []
[]

>>> [1] + [2] + [3] + [4]
[1, 2, 3, 4]

>>> [1,2,3] + [4]
[1, 2, 3, 4]

>>> [1,2,3] + [] + [4]
[1, 2, 3, 4]
```

¿¿Concatenar listas con tuplas??

- Conversión de tipos: así como `a=int(b)` convierte `b` en un `int` (sea `b` un `int`, `float`, `str`), podemos convertir una secuencia de un tipo en otra secuencia de otro tipo:

```
A = [1, 2.0, "iii", "cuatro"]                    # una lista
B = (5, 6.0, 'vii', 'eight')                     # una tupla

L = A + list(B)                  # concatena A con B-como-lista
M = tuple(A) + B                 # concatena A-como-tupla con B

X = list(A) + list(A)                # A ya era lista, se clona
```

Acceder a elementos según su índice

True	False	10	20	-5.5	'yup'	0	0	'nope'
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

```
x = [True, False, 10, 20, -5.5, 'yup', 0, 0, 'nope']
```

Python nos deja acceder a los elementos de una lista de manera sencilla

- Basta indicar su índice o posición en la lista
- Funciona igual que con strings y tuplas

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Acceder a elementos según su índice

True	False	10	20	-5.5	'yup'	0	0	'nope'
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

El uso de índices es como en ***strings***. Sabiendo esto, qué entrega:

- `len(x)` -->
- `x[ 1 ]` -->
- `x[ 6 ]` -->

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Acceder a elementos según su índice

True	False	10	20	-5.5	'yup'	0	0	'nope'
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

El uso de índices es como en ***strings***. Sabiendo esto, qué entrega:

- `len(x)` --> **`9`**
- `x[ 1 ]` --> **`False`**
- `x[ 6 ]` --> **`0`**

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Acceder a elementos según su índice

True	False	10	20	-5.5	'yup'	0	0	'nope'
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

El uso de índices es como en ***strings***. Sabiendo esto, qué entrega:

- `x[ -4 ]` -->
- `x[ -1 ]` -->
- `x[ -len(x) ]` -->

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Acceder a elementos según su índice

True	False	10	20	-5.5	'yup'	0	0	'nope'
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

El uso de índices es como en ***strings***. Sabiendo esto, qué entrega:

- `x[ -4 ]` --> **`'yup'`**
- `x[ -1 ]` --> **`'nope'`**
- `x[ -len(x) ]` --> **`True`**

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Rebanadas (*slices*) de listas

True	False	10	20	-5.5	'yup'	0	0	'nope'
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

El uso de *slices* es como en ***strings***. Sabiendo esto, qué entrega:

- `x[ 0 : 3 ]` -->
- `x[ 5 : 9 ]` -->
- `x[ -7:-2 ]` -->

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Rebanadas (*slices*) de listas

True	False	10	20	-5.5	'yup'	0	0	'nope'
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

El uso de *slices* es como en ***strings***. Sabiendo esto, qué entrega:

- `x[ 0 : 3 ]` --> **`[True, False, 10]`**
- `x[ 5 : 9 ]` --> **`['yup', 0, 0, 'nope']`**
- `x[ -7:-2 ]` --> **`[10, 20, -5.5, 'yup', 0]`**

Rebanadas (*slices*) de listas

True	False	10	20	-5.5	'yup'	0	0	'nope'
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

El uso de *slices* es como en ***strings***. Sabiendo esto, qué entrega:

- `x[ 4 :-4 ]` -->
- `x[ -3:   ]` -->
- `x[   : 3 ]` -->

Rebanadas (*slices*) de listas

True	False	10	20	-5.5	'yup'	0	0	'nope'
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

El uso de *slices* es como en ***strings***. Sabiendo esto, qué entrega:

- `x[ 4 :-4 ]` --> **`[-5.5]`**
- `x[ -3:   ]` --> **`[0, 0, 'nope']`**
- `x[   : 3 ]` --> **`[True, False, 10]`**

Recorriendo usando índices

- Podemos recorrer los elementos de una lista usando índices

- Ejemplo con **for**:

```
for i in range(len(L)):
    print( i, "->", L[i] )
```

- Ejemplo con **while**:

```
i = 0
while i < len(L):
    print( i, "->", L[i] )
    i = i+1
```

Con `L = [True,False,0,1,2,3,4,5,"yeah"]`, output:

```
0 -> True
1 -> False
2 -> 0
3 -> 1
4 -> 2
5 -> 3
6 -> 4
7 -> 5
8 -> yeah
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Caso especial: for sobre listas

- La instrucción **for** también puede iterar sobre listas, obteniendo directamente cada elemento de la lista en cuestión
  - Igual que con strings y tuplas

- Código de ejemplo:

```
L = [True,False,0,1,2,3, 4,5,"yeah"]

for c in L:
    print(c)
```

Output:

```
True
False
0
1
2
3
4
5
yeah
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Podemos detectar elementos (in)

- La instrucción **in** detecta pertenencia
- **in** entrega **True**/**False** (booleano)
- Similarmente, está la instrucción **not in**, que es la negación de **in**

```
>>> L = [10,0.5,'txt']
>>> 10 in L
True
>>> 'txt' in L
True
>>> 0 in L
False
>>> 10 not in L
False
>>> 'txt' not in L
False
>>> 0 not in L
True
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

No podemos detectar sublistas (in)

- Una sublista es una porción (rebanada, slice) de una lista
  - Ej. [10,5] es sublista de [1,10,5,8]

- La instrucción **in** *no detecta* sublistas
  - ...ni subtuplas

```
>>> L = [10,0.5,'txt']

>>> [10,0.5] in L
False
>>> [10,0.5] == L[:2]
True

>>> [10] in L
False
>>> [10] == L[:1]
True
```

Contar elementos (count)

x.count(u)

- Entrega la cantidad de veces que aparece el elemento *u* dentro de *x*
- Si el elemento no fue encontrado, entrega 0

```
>>> x = [5,-5,0,0.5,"python",0,0]
>>> x.count(0)
3
>>> x.count(5)
1
>>> x.count(-5)
1
>>> x.count("python")
1
>>> x.count(10)
0
>>> x.count("PYTHON")
0
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Encontrar elementos (index)

x.index(u)

- Encuentra el *primer* índice en el cual se encuentra el elemento ***u***, o **falla**

```
>>> x = [5,-5,0,0.5,"python",0,0]
>>> x.index(0)
2
>>> x.index(5)
0
>>> x.index(-5)
1
>>> x.index("python")
4

>>> x.index(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.index(x): x not in list
>>> x.index("PYTHON")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.index(x): x not in list
```

Mutación de listas
(para cátedra 30)

Lo siguiente no aplica a tuplas

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Asignar elementos (=)

L[i] = u

- Asigna el valor u a la posición (índice) *i* de *L*

```
>>> a = [1,2,3]

>>> a[0] = -1
>>> print(a)
[-1, 2, 3]

>>> a[0] = 1
>>> print(a)
[1, 2, 3]

>>> b = a
>>> b[-1] = 1000
>>> print(b)
[1, 2, 1000]
>>> print(a)
[1, 2, 1000]
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Anexar elementos (append)

L.append(u)

- Anexa el valor ***u*** al final de la lista *L*, como un elemento más

```
>>> a = [1,2,3]

>>> a.append(-1)
>>> print(a)
[1, 2, 3, -1]

>>> a.append([0,5])
>>> print(a)
[1, 2, 3, -1, [0, 5]]

>>> b = a
>>> b.append("!!!")
>>> print(b)
[1, 2, 3, -1, [0, 5], '!!!']
>>> print(a)
[1, 2, 3, -1, [0, 5], '!!!']
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Insertar elementos (insert)

L.insert(i, u)

- Inserta el valor *u* en el índice *i* de la lista *L*, desplazando los índices de los elementos posteriores en la lista en **+1** lugar

```
>>> L = [0,1,2,3,4]
>>> print(L)
[0, 1, 2, 3, 4]

>>> L.insert(0,"ari")
>>> print(L)
['ari', 0, 1, 2, 3, 4]

>>> L.insert(3,"gato")
>>> print(L)
['ari', 0, 1, 'gato', 2, 3, 4]

>>> L.insert(-1,"miau")
>>> print(L)
['ari',0, 1,'gato',2, 3,'miau',
4]
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Remover elementos (pop)

L.pop(i)

- Remueve el elemento en el índice *i* y lo retorna
- Si *i* no se entrega, se asume *i* = -1 (remover el último elemento)
- Los índices de los elementos posteriores cambian en -1

```
>>> L = ['cat', 'dog', 'bird']
>>> print( L )
['cat', 'dog', 'bird']

>>> a = L.pop( 0 )
>>> print( a )
cat
>>> print( L )
['dog', 'bird']

>>> a = L.pop( -1 )
>>> print( a )
bird
>>> print( L )
['dog']
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Extender una lista con otra (extend)

L.extend(M)

- Toma los elementos de *M* y los anexa al final de *L*
- *M* puede ser lista, tupla, string, u otra secuencia de valores
  - Si *M* es lista o tupla: se anexan los elementos a *L*
  - Si *M* es string: se anexa cada caracter como elemento a *L*

```
>>> L = [1,2]
>>> print(L)
[1, 2]

>>> L.extend([3,4])
>>> print(L)
[1, 2, 3, 4]

>>> L.extend((5,6))
>>> print(L)
[1, 2, 3, 4, 5, 6]

>>> L.extend("78")
>>> print(L)
[1, 2, 3, 4, 5, 6, '7', '8' ]
```