COL1000
Introduction to Programming

Priyanka Golia

Most (if not all) of the content is borrowed from Prof. Subodh Kumar's slides

Quiz

L =[[10,20],30,40]

allowed or syntax error Allowed

L = [[10,20],30,40]

print(*L*[0][1]) 20

Slicing

Slicing extracts a portion (subsequence) of a list, string, or tuple.

Syntax: sequence[start:end:step]

Just like range, start is inclusive and end is exclusive

Default step size is 1.

String and tuple are immutable objects — slicing returns a new object.
lists are mutable.

Important points to remember—
1. If Start > length, returns empty list.

2. If End > length, stops at the last element

Slicing never raises an error if indices are out of bounds.

3. Slicing can handle negative indices — counts from end.

4. Negative step reverses direction.

Slicing

$L = [10,20,30,40,50]$

#index 0,1,2,3,4

Slice	Result	Explanation
L[0:3]	[10, 20, 30]	From index 0 to 2 (end=3 excluded).
L[:3]	[10, 20, 30]	Omit start → from beginning.
L[2:]	[30, 40, 50]	Omit end → till last element.
L[:]	[10, 20, 30, 40, 50]	Full shallow copy.
L[-1]	50	Last element (single index).
L[-2:]	[40, 50]	Last 2 elements.
L[:-2]	[10, 20, 30]	All except last 2.
L[-4:-1]	[20, 30, 40]	From 2nd to last, stop before last.
L[::2]	[10, 30, 50]	Step 2 → every second element.
L[1::2]	[20, 40]	Start at index 1, step 2.
L[::-1]	[50, 40, 30, 20, 10]	Step –1 → full reversal.
L[3:0:-1]	[40, 30, 20]	Reverse slice: start=3, go backwards to index >0.
L[4:1:-2]	[50, 30]	Reverse with step –2.
L[10:]	[]	Start > length → empty list.
L[:10]	[10, 20, 30, 40, 50]	End > length → stops at last element.

Slicing

```
s = "PYTHON"
# indices:  0 1 2 3 4 5
# letters:  P Y T H O N
```

Slice	Result	Explanation
s[0:3]	"PYT"	Start=0, stop before 3 → indices [0,1,2].
s[:4]	"PYTH"	From beginning up to index 3.
s[2:]	"THON"	From index 2 to end.
s[:]	"PYTHON"	Full shallow copy.
s[-1]	"N"	Last character.
s[-3:]	"HON"	Last 3 characters.
s[:-2]	"PYTH"	All except last 2.
s[-5:-2]	"YTH"	From index -5 to -3.
s[::2]	"PTO"	Every 2nd character.
s[::-1]	"NOHTYP"	Reverse string.
s[10:]	""	Start > length → empty string.
s[:10]	"PYTHON"	Stop beyond length → goes till end.

Slicing: List vs String

```
L = [1,2,3,4]

L[1 : 3] = [20,30]

Print(L)  # Mutable object — L = [1,20,30,4]
```

```
s = "hello"

s[1 : 3] = "xx"
# TypeError:’str' object does not support item assignment

s = s[:1] + "xx" + s[3:]  →  Creates a new object.

print(s)   # "hxxlo"
```

Methods to create a list

Using Square Brackets [] — The simplest and most common method.

```
L = [1,2,3,4]
Print(L) #L=[1,2,3,4]
```

Using the list() — Converts other iterables (strings, tuples, sets, ranges) into a list.

```
print(list("hello"))#['h','e','l','l','o']
print(list((1, 2, 3)))# [1, 2, 3]
print(list(range(5))) # [0, 1, 2, 3, 4]
```

From Empty List — Start with an empty list and add elements later.

```
L = []
L.append(10)
L.append(20)
print(L)#[10, 20]
```

```
L = []
L.extend([10,20])
print(L)# [10, 20]
```

Extend to add multiple elements.
Notice the syntax

Using split() (from a string) — Split string into list of substrings.

```
text = "apple,banana,cherry"
fruits = text.split(",")
print(fruits)   # ['apple', 'banana', 'cherry']
```

```
L = [2]*4
print(L)# [2,2,2,2]
```

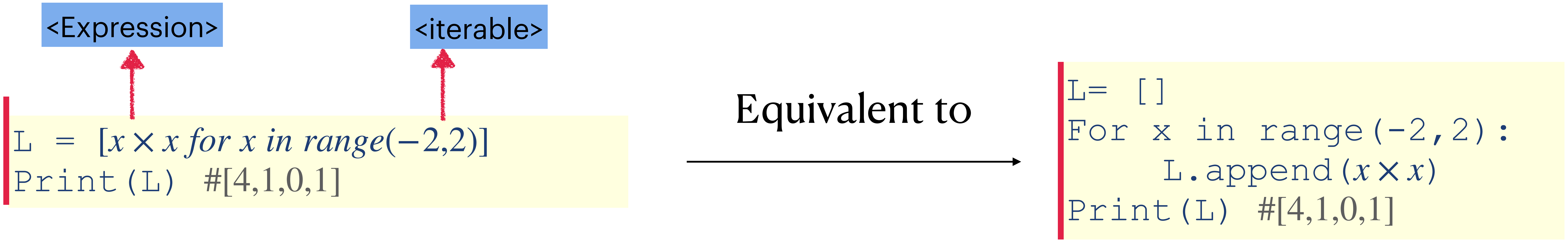
Creation via repetition is allowed.

Methods to create a list

Using **List Comprehension**: Just like one can iterate over elements in a list, one can have a loop create elements in a list

List = [<expression> for <name> in <iterable> if <condition>]

<expression>: what you put into the new list, can include complex expression including if-else
for <name> in <iterable>: where items come from. (Could be nested loops)
if <condition> (optional): keep only items that pass the test.



```
L  = [x × x for x in range(−2,2)]
Print(L)  #[4,1,0,1]
```

Equivalent to

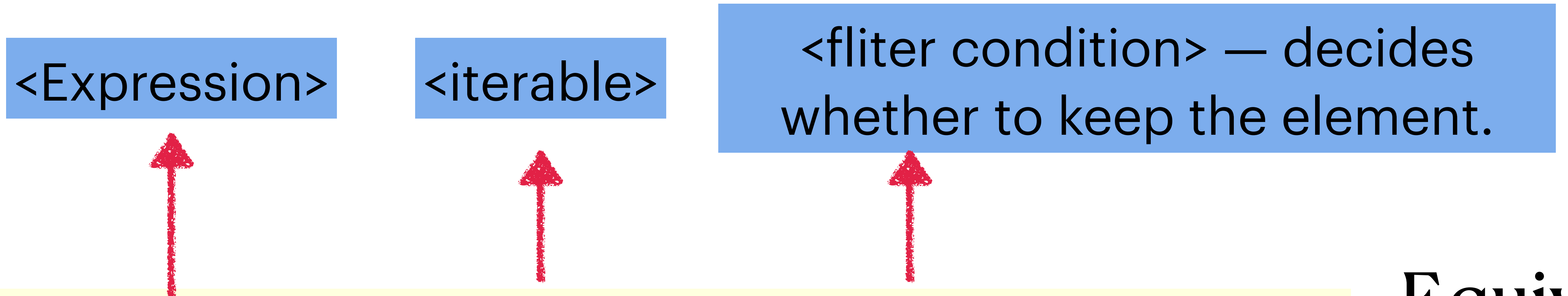
```
L= []
For x in range(-2,2):
      L.append(x × x)
Print(L)  #[4,1,0,1]
```

Methods to create a list

Using **List Comprehension**: Just like one can iterate over elements in a list, one can have a loop create elements in a list

List = [<expression> for <name> in <iterable> if <condition>]

<expression>: what you put into the new list, can include complex expression including if-else
for <name> in <iterable>: where items come from. (Could be nested loops)
if <condition> (optional): keep only items that pass the test.



<Expression>

<iterable>

<fliter condition> — decides whether to keep the element.

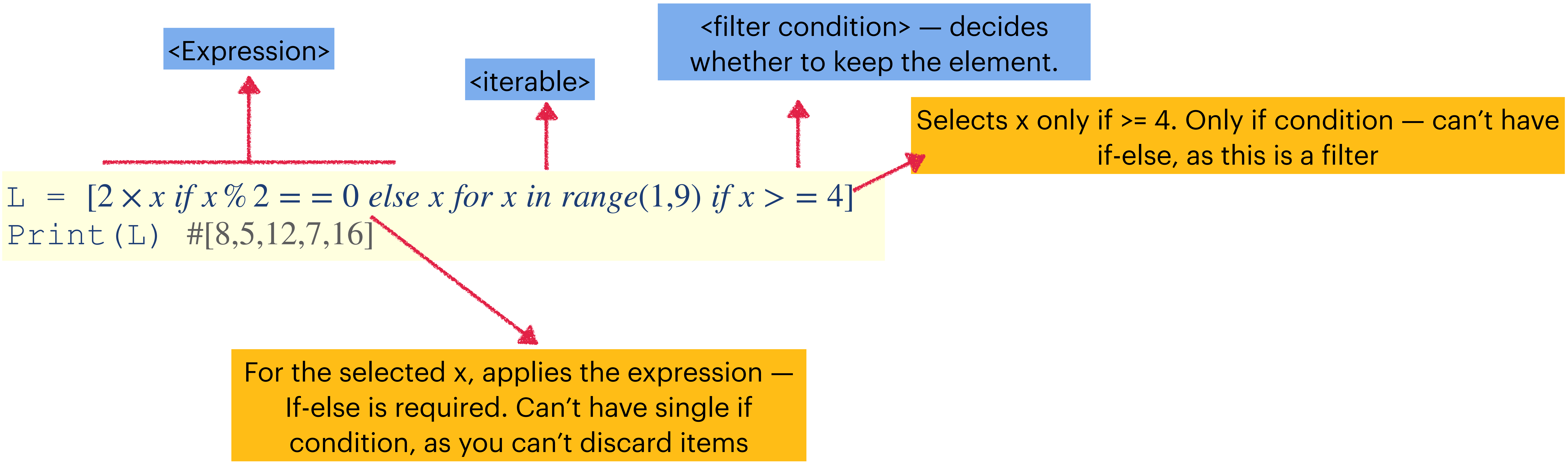
```
L  =  [x for x in range(1,6) if x % 2 = = 0]
Print(L)  #[2,4]
```

Equivalent to →

```
L=  []
For x in range(-2,2):
     If x % 2 == 0:
          L.append(x)
Print(L)  #[2,4]
```

Methods to create a list

Using **List Comprehension**: Just like one can iterate over elements in a list, one can have a loop create elements in a list



<Expression>

<iterable>

<filter condition> — decides whether to keep the element.

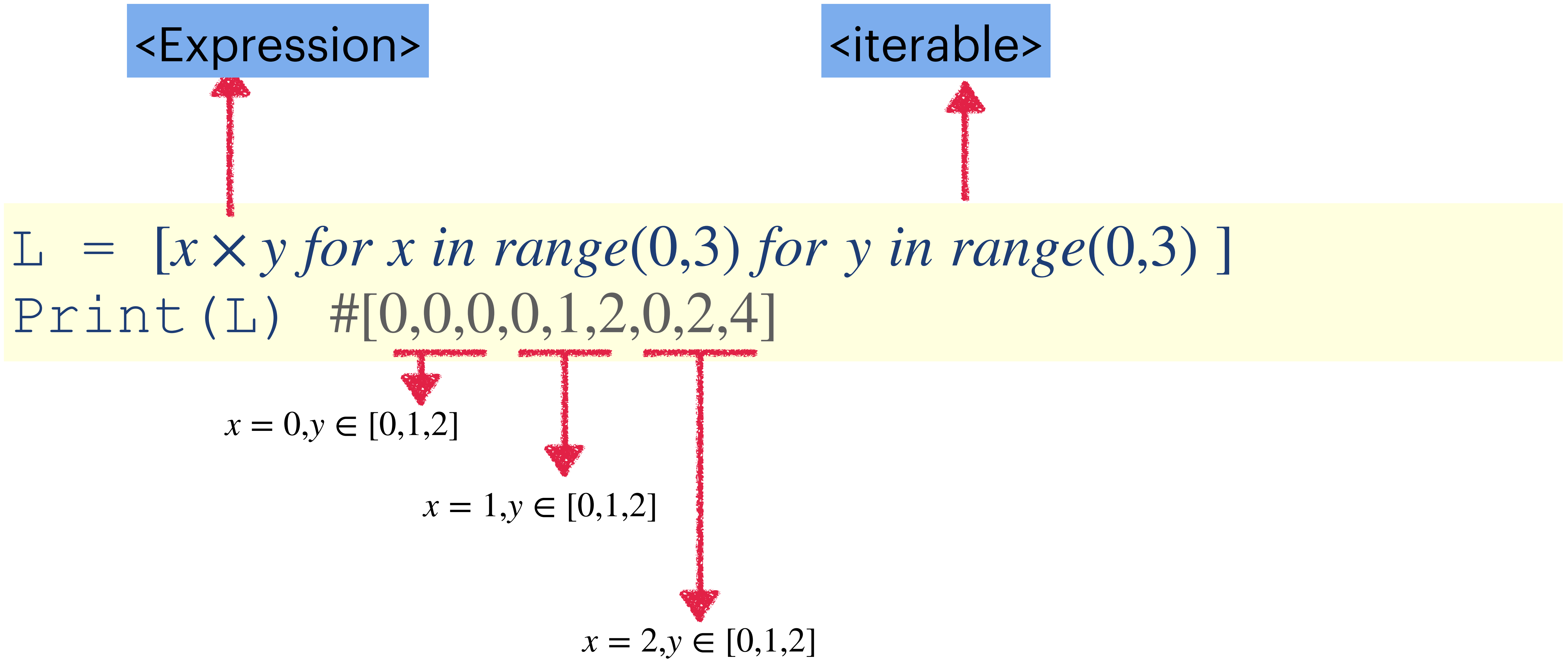
Selects x only if >= 4. Only if condition — can’t have if-else, as this is a filter

```
L = [2 × x if x % 2 = = 0 else x for x in range(1,9) if x > = 4]
Print(L) #[8,5,12,7,16]
```

For the selected x, applies the expression — If-else is required. Can’t have single if condition, as you can’t discard items

Methods to create a list

Using **List Comprehension**: Just like one can iterate over elements in a list, one can have a loop create elements in a list



<Expression>

<iterable>

L = $[x \times y \; for \; x \; in \; range(0,3) \; for \; y \; in \; range(0,3) \;]$

Print(L) #[0,0,0,0,1,2,0,2,4]

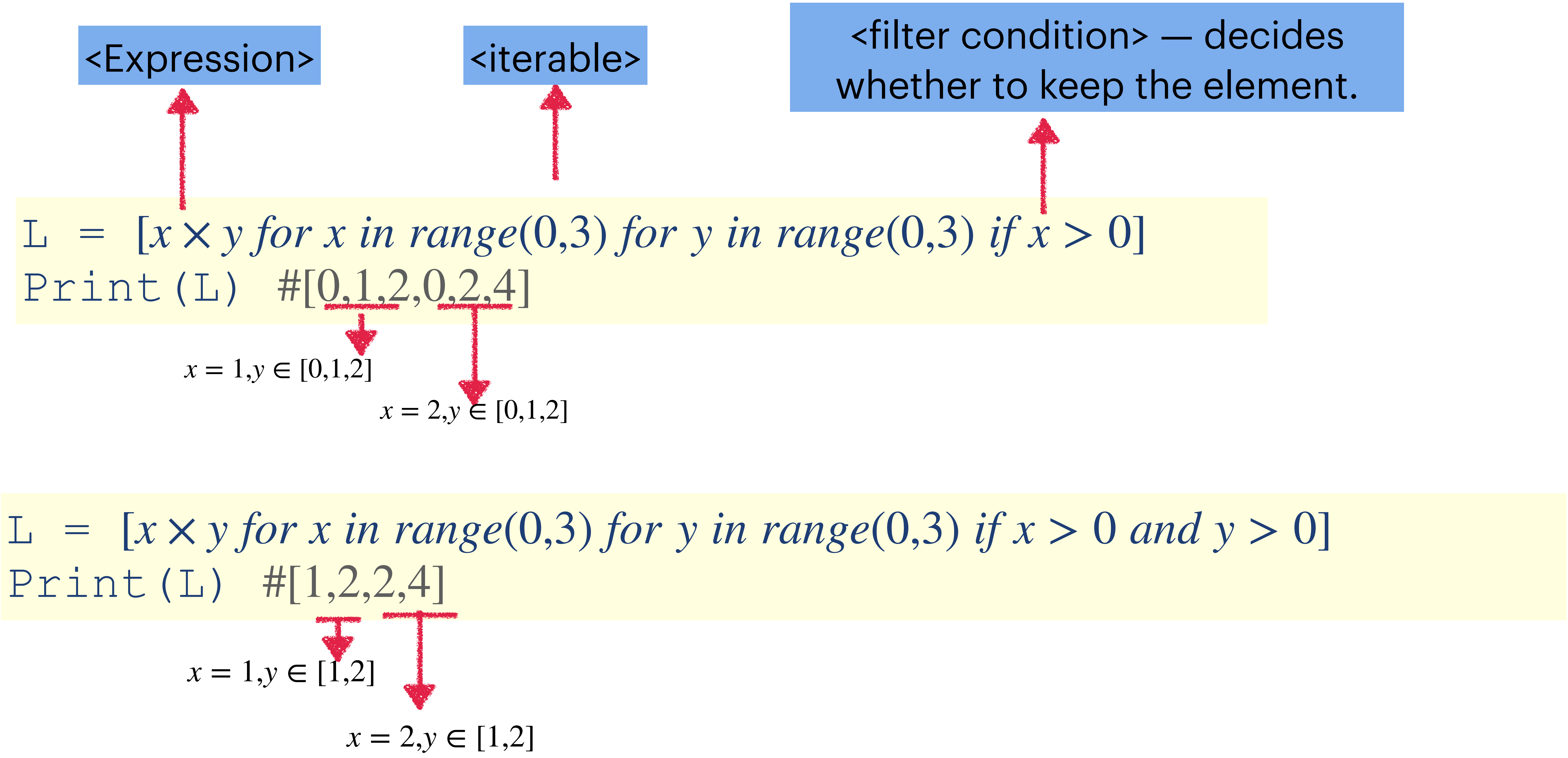
$x = 0, y \in [0,1,2]$

$x = 1, y \in [0,1,2]$

$x = 2, y \in [0,1,2]$

Methods to create a list

Using **List Comprehension**: Just like one can iterate over elements in a list, one can have a loop create elements in a list



<Expression>

<iterable>

<filter condition> — decides whether to keep the element.

L = $[x \times y \; for \; x \; in \; range(0,3) \; for \; y \; in \; range(0,3) \; if \; x > 0]$

Print(L) #$[0,\underbrace{1,2,0}_{x=1, y \in [0,1,2]},\underbrace{2,4}_{x=2, y \in [0,1,2]}]$

L = $[x \times y \; for \; x \; in \; range(0,3) \; for \; y \; in \; range(0,3) \; if \; x > 0 \; and \; y > 0]$

Print(L) #$[\underbrace{1,2}_{x=1, y \in [1,2]},\underbrace{2,4}_{x=2, y \in [1,2]}]$

Methods to create a list

Using **List Comprehension**: Just like one can iterate over elements in a list, one can have a loop create elements in a list

```
L1 = [4,9,16]
Import math
L = [math.sqrt(x) for x in L1 ]
Print(L) #[2,3,4]
```

Applies operation to every element of a list

<Expression>

<iterable>

For every row, creates a lists.

```
mat = [[col for col in range(0,3)] for row in range(0,3)]
Print(mat) #[[0,1,2], [0,1,2], [0,1,2]]
```

It is a 3 x 3 matrix (list of lists).

Tuples

mytuple = ("apple", "banana", "cherry")   mytuple= ("abc", 34, True, 40.0, "male")

Syntax: Lists use square brackets [], whereas tuples use parentheses ()

Just like list, tuple has ordinal constraints —i.e, ordering and comparisons constraints.
Can be indexed, iterated, has len()

Both list and tuples supports lexicographical comparison.

```
(1, 2, 3) < (1, 3, 0)  # True, compares first differing element
[1, 2, 3] < [1, 3, 0]  # True, same behavior as tuples
```

Tuples are immutable, however list are mutable.

Memory for tuples is fixed once created, therefore, often faster to access.

Lists have more built-in methods (e.g., append, remove) compared to tuples, which have fewer (e.g., count, index).

Matrix in Python — list of lists.

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

Notice the comma (,) between lists.
Each inner list is an element of outer list.

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

- Number of outer list elements — number of rows.
  That is, len(matrix) is the number of rows.
- Now, each row is a list of elements (inner lists).
  That is, len(matrix[0]) is number of column.
- To access the i^{th} row — matrix[i].
- To access the element at i^{th} row and j^{th} column — matrix[i][j]

Matrix in Python — list of lists.

If you have a flat list and want to reshape it into an n by m matrix (list of lists)

a =[1,2,3,4,5,6,7,8,9,10,11,12] → [[1,2,3,4], [5,6,7,8], [9,10,11,12]]

```
n, m = 3,4  # desired matrix dimensions: 3 rows, 4 columns
matrix = [a[i : i + m] for i in range(0,n × m, m)]
print(matrix)
# Output: [[1,2,3,4], [5,6,7,8], [9,10,11,12]]
```