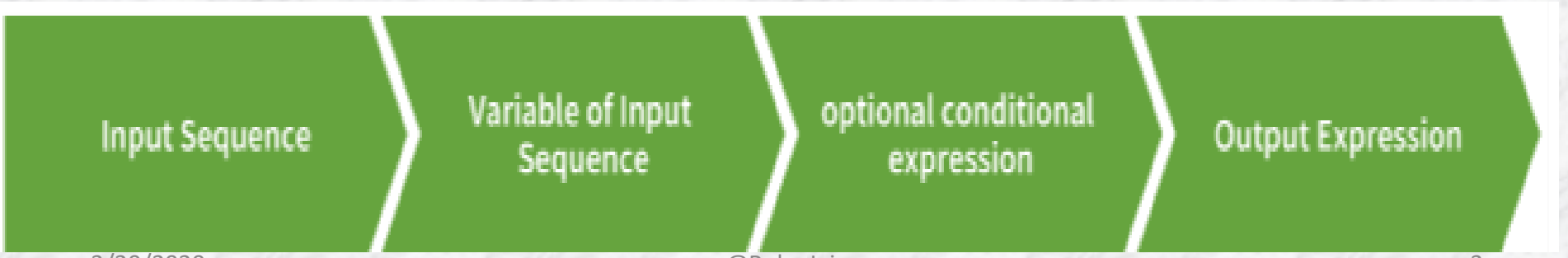
Comprehension

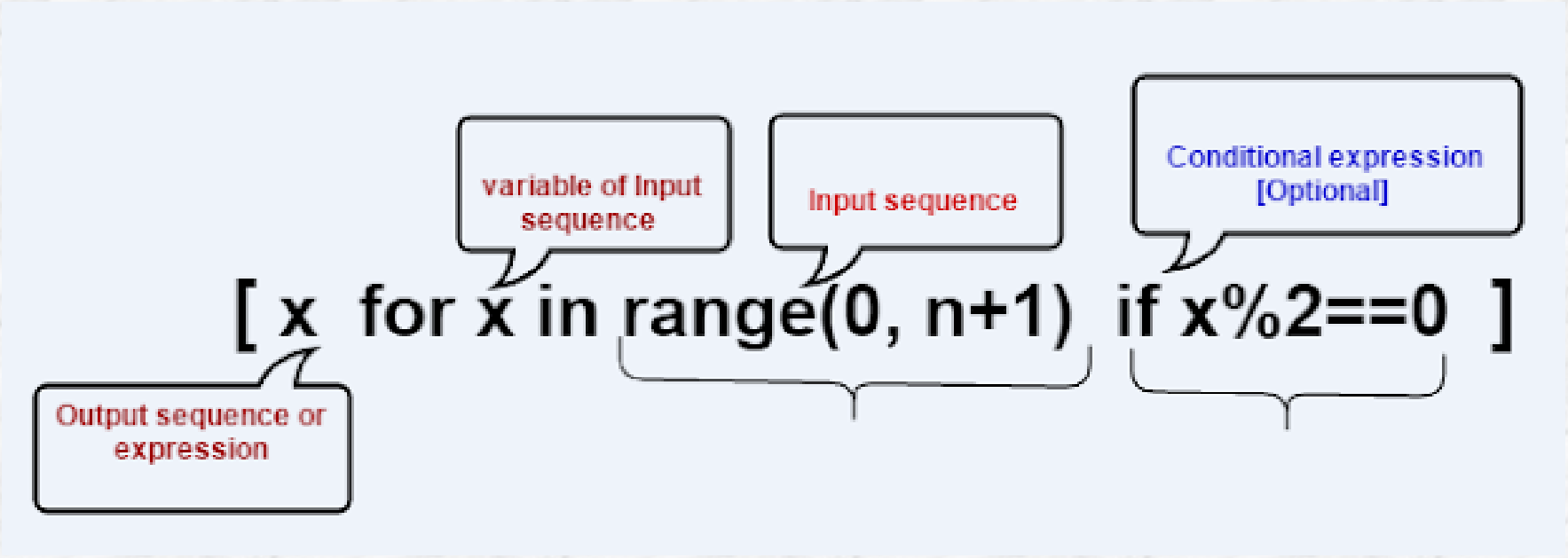
Types of Comprehensions in Python

List Comprehensions [ ]

- List comprehension is a way to define and create lists in Python in a concise way. In most cases, list comprehensions let us create lists in a single line of code without worrying about initializing lists or setting up loops.
- A list comprehension consists of the following parts:



values = [**x for** x **in** range(**0**, n+**1**) **if** x%**2**==**0**]



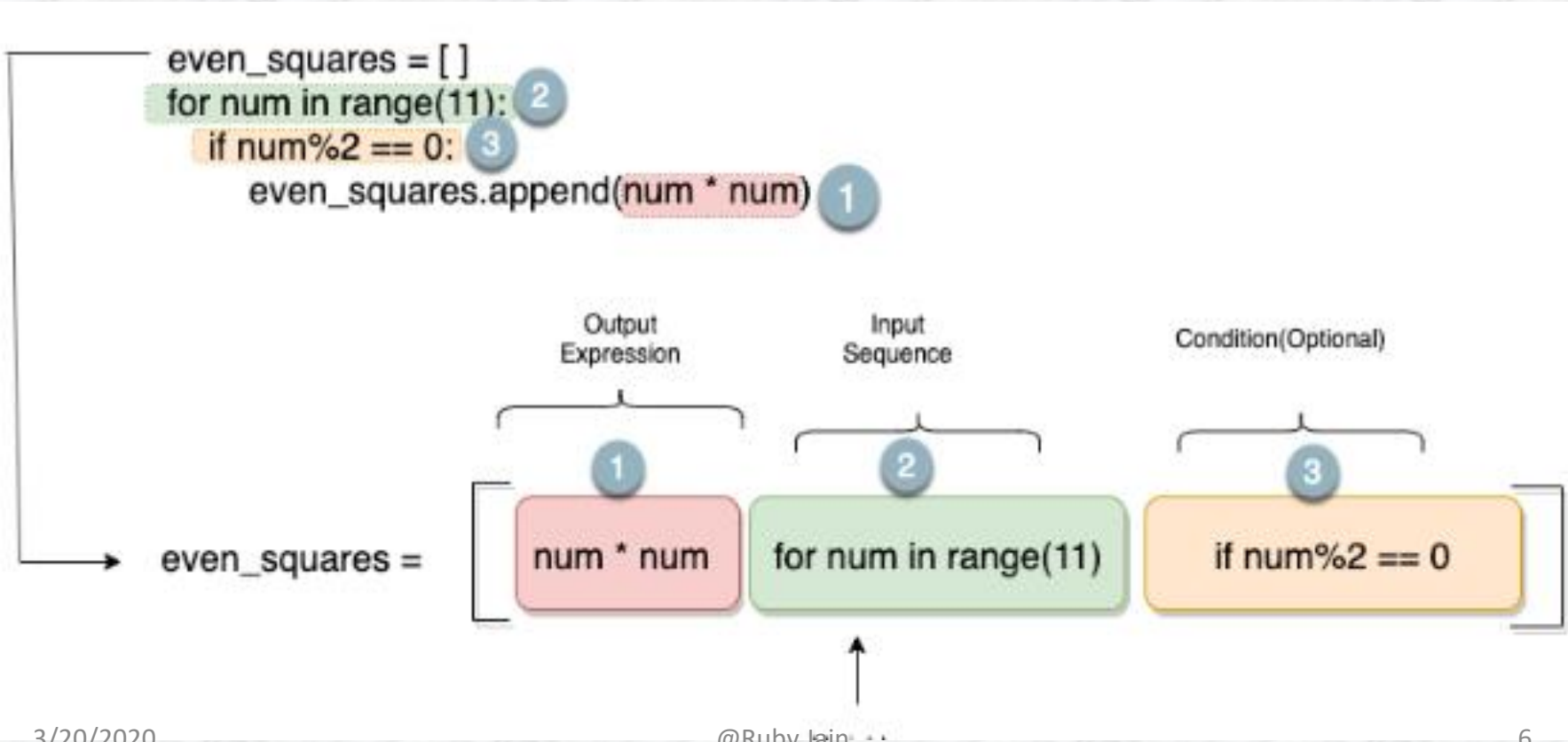
Example

```
even_squares = [num * num for num in range(11)
    if num%2 == 0]

even_squares

#[0, 4, 16, 36, 64, 100]
```

```
Even_squares=[]
for num in range(11):
        if num%2 == 0:
                even_squares.append(num * num)
```



List Comprehensions with strings

- Converting lowercase letters in a string to uppercase.

```
colors = ["pink", "white", "blue", "black", "purple"]
[color.upper() for color in colors]
```

Nested List Comprehensions [[ ]]

- List comprehensions can also be nested to create complicated lists. For instance, we can create a matrix using only list comprehensions.

```
matrix = [[j * j+i for j in range(3)] for i in range(3)]
matrix
#[[0, 1, 4], [1, 2, 5], [2, 3, 6]]
```

List Comprehensions with tuples

- If the expression contains a tuple (e.g. the (x, y) , it must be parenthesized.
- Example:

height_in_cms =
[('Tom',183),('Daisy',171),('Margaret',179),('Michael',190),('Nick',165)]

height_in_feet =
[(height[0],round(height[1]*0.0328,1)) for height in height_in_cms]

height_in_feet

Set Comprehensions { }

A set comprehension is similar to a list comprehension but returns a set instead of a list. The syntax is slightly different in the sense that we use curly brackets instead of square brackets to create a set.

```
names = [ 'Arnold', 'BILL', 'alice', 'arnold', 'MARY', 'J', 'BIll'
  ,'maRy']
{name.capitalize() for name in names if len(name) > 1}

#{'Alice', 'Arnold', 'Bill', 'Mary'}
```

Dictionary Comprehensions { }

Dictionary comprehensions are used when the input is in the form of a dictionary or a key: value pair. For instance, consider a dictionary where the keys represent characters, and the values denote the number of times these characters appear in a corpus.

Generator Expressions ( )

- The syntax and the way of working of generator expressions are precisely like a list comprehension except that they use round brackets instead of square ones. Let's say we want to calculate the sum of squares of the first ten natural numbers.
- Unlike list comprehensions, a generator expression doesn't return a list but a generator object. To get the result, we can use the above expression with the sum function.