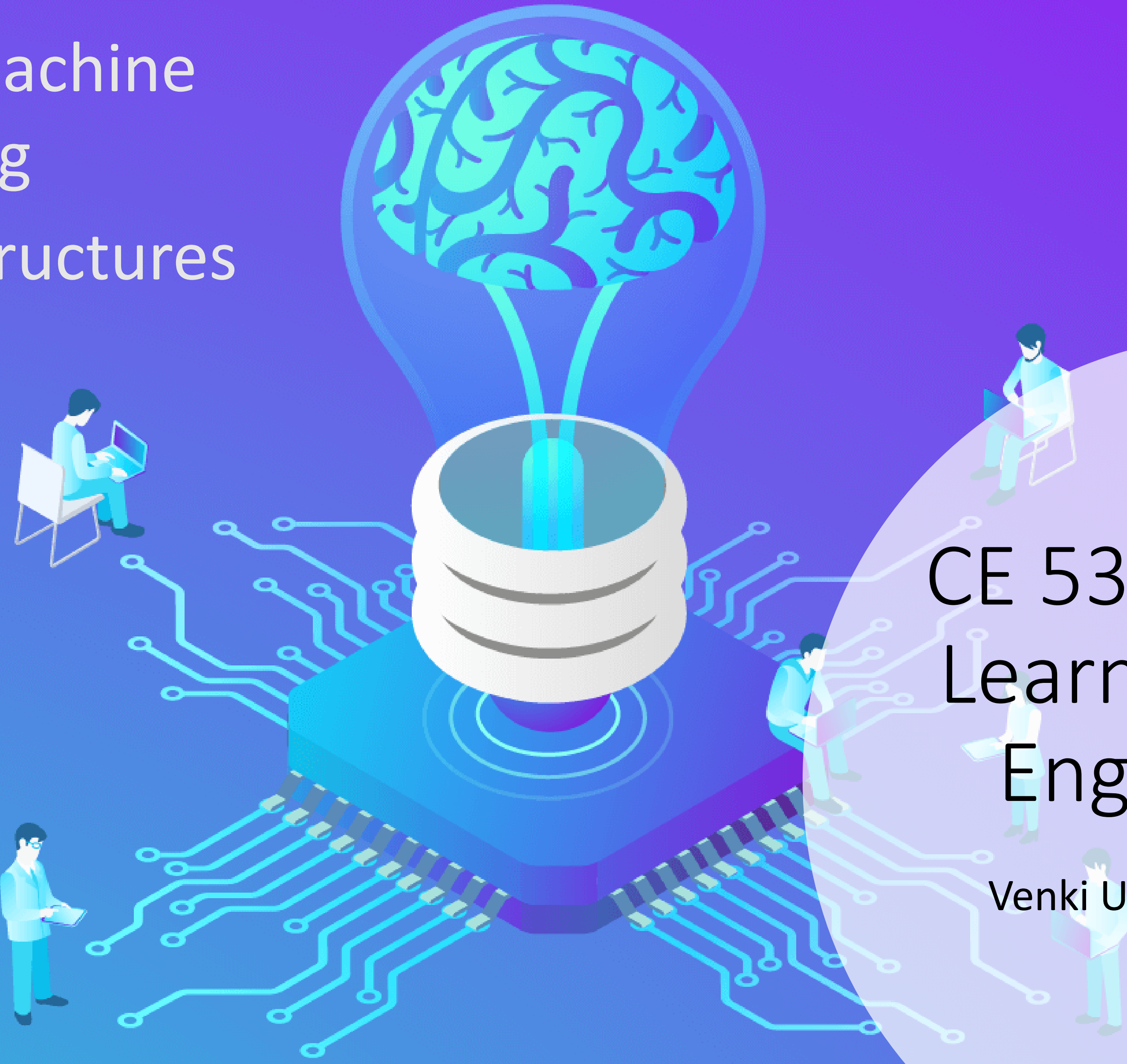
Python for Machine Learning

Native Data Structures



CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

Recap and Goals

- Installed Python and Anaconda Environments
- Introduction to Python
  - Setting working directory
  - Adding comment lines
    - Docstrings

Goal of this module is to explore Basic Python Data Structures

Data Structures

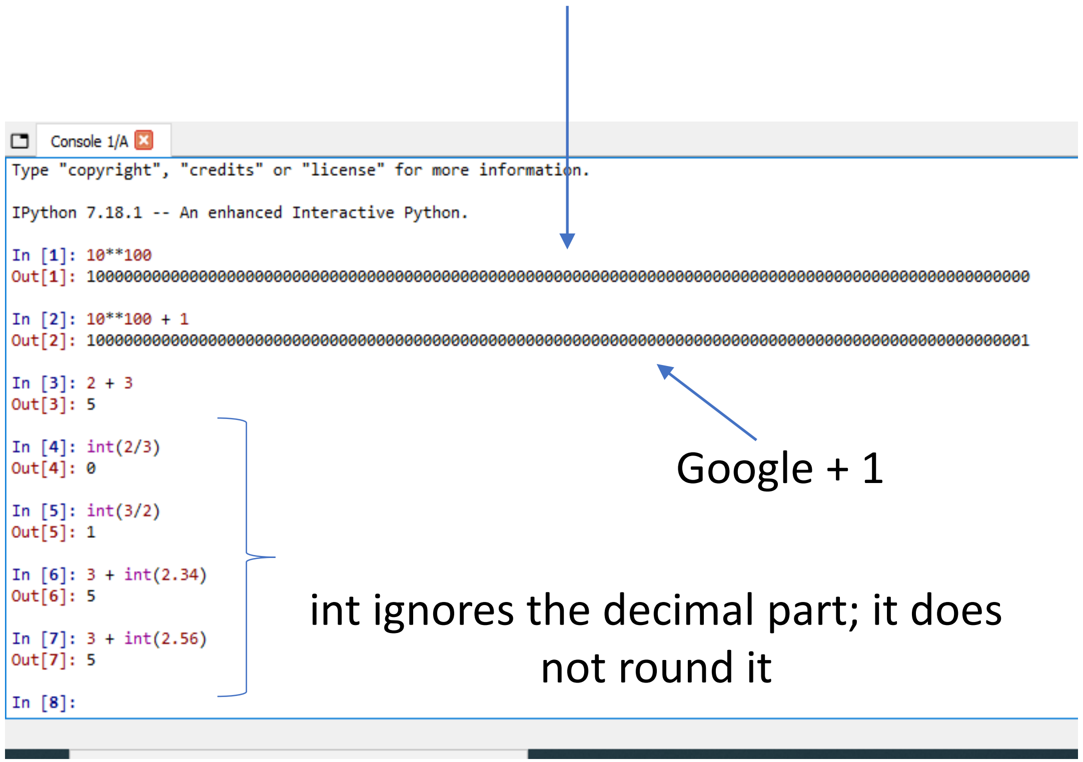
- Data structures are fundamental to programming
- Data structures tell us what type of data can be stored and used within a programming language
- Understanding data structures is therefore important to manipulate data types within Python
- Python is a lean language
  - Has a few essential data structures
  - These can be enhanced via libraries
- Python has some unique data structures as well
  - Not found in other languages

Here we shall explore some basic data structures inherent to python

Integers

- Integers are a fundamental data structure in Python
- Python is unique in that it supports “Infinite Integers”
  - Very large integers can be specified exactly
  - This feature is unique to python
    - Possible to develop such functionality in others but does not come built-in
  - TweetID (every tweet on Twitter has a unique ID)
    - This ID is an integer
    - Python can decode this TweetID very easily due to its infinite integer capability

Google



```
Type "copyright", "credits" or "license" for more information.

IPython 7.18.1 -- An enhanced Interactive Python.

In [1]: 10**100
Out[1]: 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

In [2]: 10**100 + 1
Out[2]: 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

In [3]: 2 + 3
Out[3]: 5

In [4]: int(2/3)
Out[4]: 0

In [5]: int(3/2)
Out[5]: 1

In [6]: 3 + int(2.34)
Out[6]: 5

In [7]: 3 + int(2.56)
Out[7]: 5

In [8]:
```

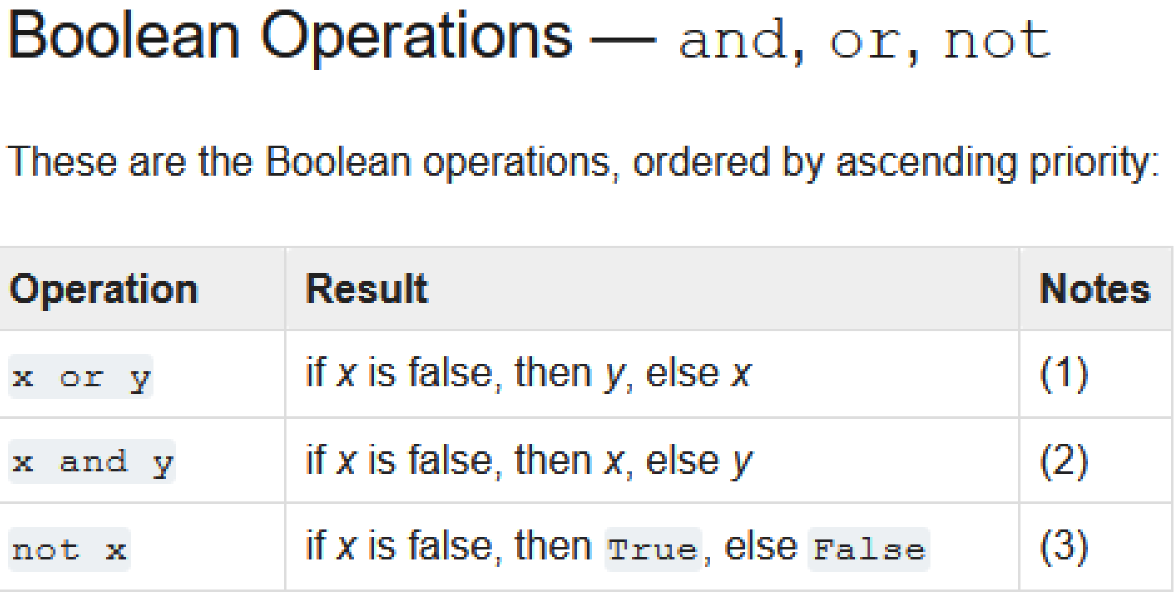
Google + 1

int ignores the decimal part; it does not round it

A Google (or Googol) is a large number 10^100 – Python can perform precise calculations even with this number

Boolean Data Type

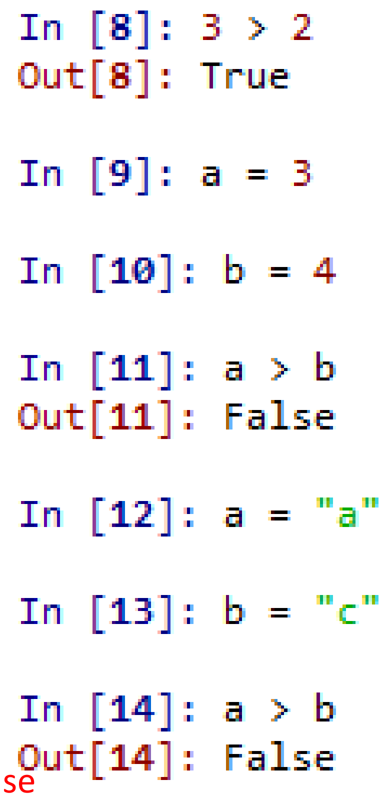
- Boolean operators are used for comparison
- Booleans are a special type of Integers



Boolean Operations — `and`, `or`, `not`

These are the Boolean operations, ordered by ascending priority:

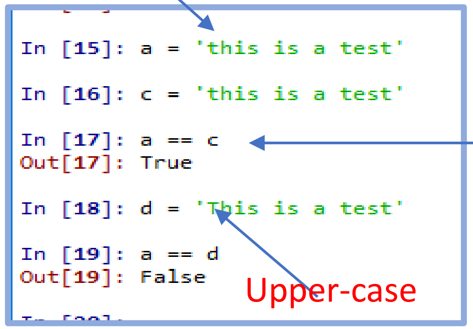
Operation	Result	Notes
`x or y`	if *x* is false, then *y*, else *x*	(1)
`x and y`	if *x* is false, then *x*, else *y*	(2)
`not x`	if *x* is false, then `True`, else `False`	(3)



```
In [8]: 3 > 2
Out[8]: True

In [9]: a = 3

In [10]: b = 4

In [11]: a > b
Out[11]: False

In [12]: a = "a"

In [13]: b = "c"

In [14]: a > b
Out[14]: False
```

Boolean Works on Strings as Well

Lower-case



```
In [15]: a = 'this is a test'

In [16]: c = 'this is a test'

In [17]: a == c
Out[17]: True

In [18]: d = 'This is a test'

In [19]: a == d
Out[19]: False
```

Upper-case

Comparison is carried out using == operator

Comparison is case-sensitive

Floating Point Numbers

- Floating point numbers are numbers with decimal places
- Python now allows operations between integer and floating-point numbers
  - Integers are promoted to Floating Point Numbers
- In many programming languages floating point numbers have greater precision than integers
  - Not so in Python
  - Regardless of the programming language – The accuracy of the floating-point number deteriorates after some point
- Many languages offer “double” precision to improve the accuracy of decimals
  - This is not natively offered in python but float is similar to double in c++
  - Use **sys.float_info** command to find out the precision on your machine.
    - Need to import **sys** library before doing so

Accurate to 15 decimals

```
sys.float_info(max=1.7976931348623157e+308,
max_exp=1024, max_10_exp=308,
min=2.2250738585072014e-308, min_exp=-1021,
min_10_exp=-307, dig=15, mant_dig=53,
epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

Note **float** is part of native python **double** is part of Numpy library

Floating Point Numbers

- Floating Point numbers are only accurate to certain decimals
  - Computer uses binary operations
  - Most decimal fractions (base 10) cannot be represented as binary fractions (base 2)
- Typically not a problem as most measurement accuracy is well below these accuracy limits
- However, Boolean operations with floating point sometimes does not give correct results
  - Because of accuracy issues

Value of 1/10

Accurate to 15 decimal places

```
format(0.1,".15f")
Out[31]: '0.100000000000000'

format(0.1,".24f")
Out[32]: '0.100000000000000005551115'
```

Inaccuracy at higher digits
(floating point error)

```
a = 0.1 + 0.1 + 0.1

a == 0.3
Out[34]: False
```

Not Equal due to
floating point error

```
round(a,10) == round(0.3,10)
Out[35]: True

round(a,15) == round(0.3,15)
Out[36]: True
```

Accurate to 15 decimals

Strings

- Strings are used to read and write text
- Strings are defined by either single or double quotes
    - You can use either but have to be consistent
- You can use “’ (triple single quote) to enclose long strings
    - Say a paragraph that includes multiple statements’
    - The DocStrings are an example of this

```
a = 'This is a test'

b = "I passed!!"

a + ' and ' + b
Out[53]: 'This is a test and I passed!!'
```

Notice the space before and after “and”

Integers and floating-point numbers can be converted to strings and concatenated

```
x = str(3) # use str to change integer to string
aa = 'The value of x is: '
aa + x
Out[59]: 'The value of x is: 3'
```

Floating Point Numbers can be formatted using format statement before concatenation

```
ys = str(format(y,'.3f')) # use format to specify decimals
bb = 'The value of y to 3 decimals is: '
bb + ys
Out[64]: 'The value of y to 3 decimals is: 0.667'
```

Sequence Types

Sequence Types

- There are three basic sequence types
  - List, Tuple and Range
- Python list is like a set of ordered boxes
  - A list is denoted by square brackets []
- Lists can be appended
- Sublists can be formed from a list
- Lists can have duplicate values
- Lists are **mutable**
  - **You can change the values without creating a new list**

Numbering of elements in the list begins at zero

A list can have no (null) or simply 1 element

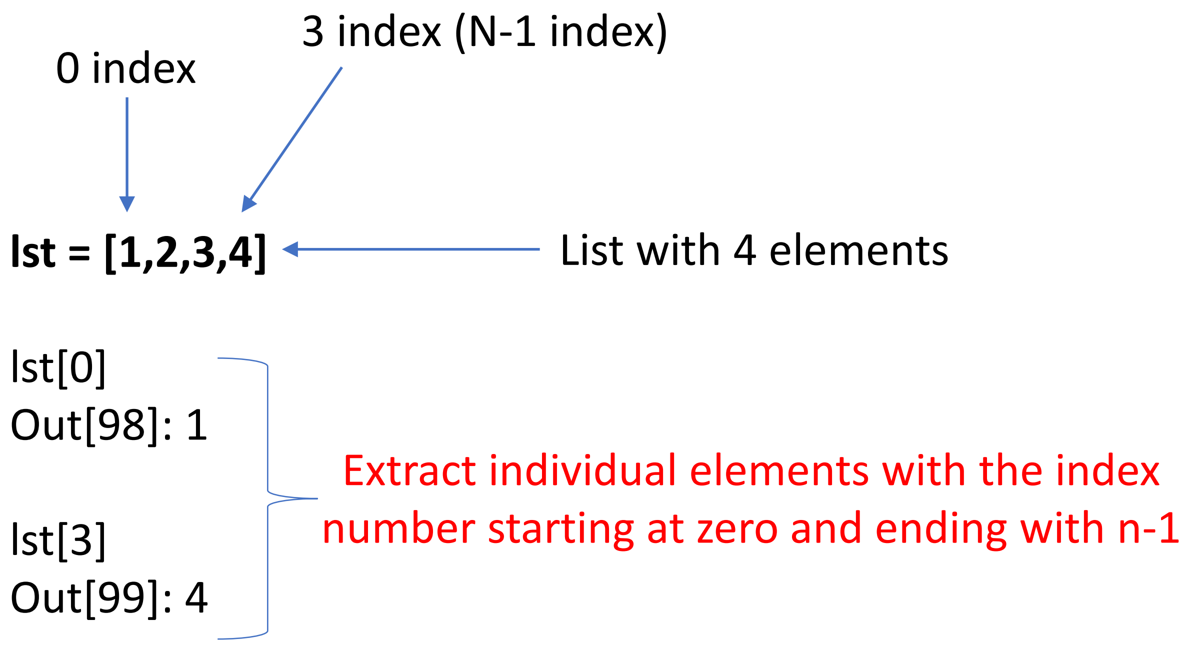
A for loop is useful to manipulate the elements of a list

Operations with a list may seem confusing, especially if you are coming from R

List Operations

A list index starts at zero and not 1

You can extract an individual element using the index



0 index

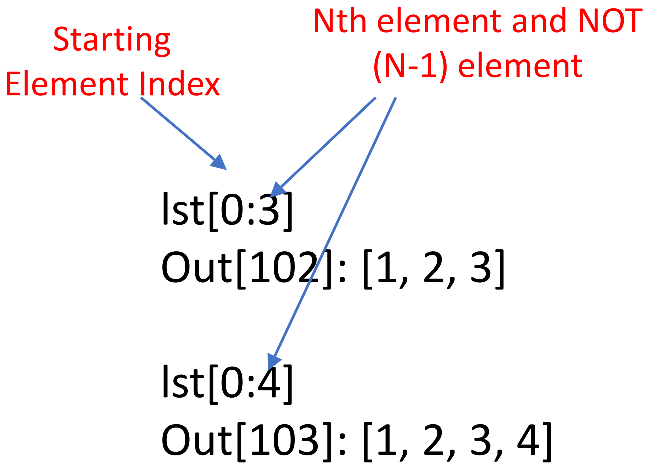
3 index (N-1 index)

lst = [1,2,3,4]

List with 4 elements

```
lst[0]
Out[98]: 1

lst[3]
Out[99]: 4
```

Extract individual elements with the index number starting at zero and ending with n-1



Starting Element Index

Nth element and NOT (N-1) element

```
lst[0:3]
Out[102]: [1, 2, 3]

lst[0:4]
Out[103]: [1, 2, 3, 4]
```

The Use of : to extract multiple elements uses starting index and **one more** the index of the element to extract and not first and last element

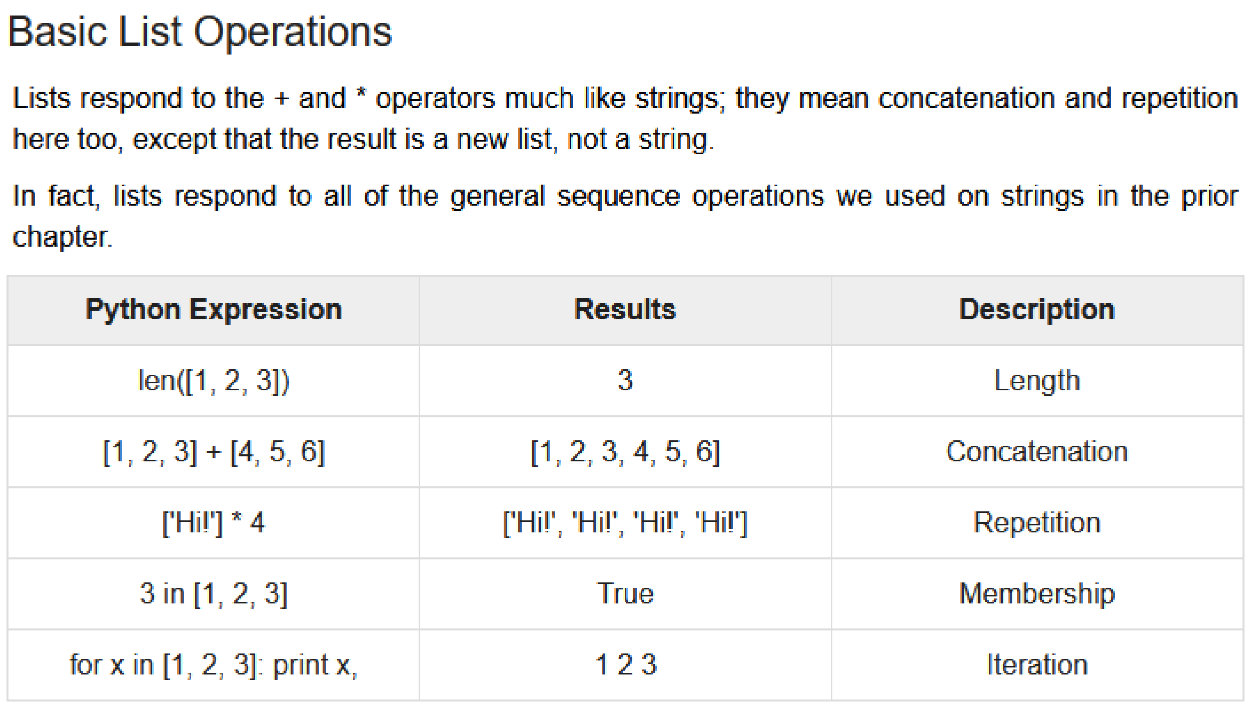
```
lst[1:2]
Out[106]: [2]

lst[1:3]
Out[107]: [2, 3]

lst[1:4]
Out[108]: [2, 3, 4]
```

Lists

- Lists can have mixed data types

- Therefore one cannot simply perform arithmetic operations on a list

- ‘*’ serves as a repetition or concatenation operator



Basic List Operations

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

In fact, lists respond to all of the general sequence operations we used on strings in the prior chapter.

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Tuples

- Tuples are similar to lists but are immutable
- Tuples use parenthesis () instead of square brackets []
- The numbering rules of tuples is the same as that of lists
  - Index starts at zero
  - There are are N-1 elements
  - When using : to extract multiple elements – Remember
    - First value : (the Nth value)
      - To extract up to and not including Nth value

Tuples are immutable which means you cannot change values from a tuple

```
zz = ('a','b','c','d')

zz[0]
Out[110]: 'a'

zz[0:4]
Out[111]: ('a', 'b', 'c', 'd')

zz[0:3]
Out[112]: ('a', 'b', 'c')
```

Notice the use of square brackets to extract values

```
tup1 = (1,2,3)
tup1[0] = 4
Traceback (most recent call last):
File "<ipython-input-116-d4bfcc7b35eb>", line 1, in <module>
  tup1[0] = 4
TypeError: 'tuple' object does not support item assignment
```

Not possible to change an element of a tuple

Tuples

- A tuple is assumed to NEVER have a single element
  - There should be a trailing comma
- If you simply put a single number or a string within parenthesis it is treated as a integer, float or a string
- To have a single element tuple – you need to add a comma after the element
  - Not necessary if there are two or more elements

Like lists tuples can also have mixed data types

```
a = (2)

type(a)
Out[129]: int
```

Integer

```
a = (2.3)

type(a)
Out[131]: float
```

Float

```
a = ('a')

type(a)
Out[133]: str
```

String

```
a = (2,)

type(a)
Out[135]: tuple
```

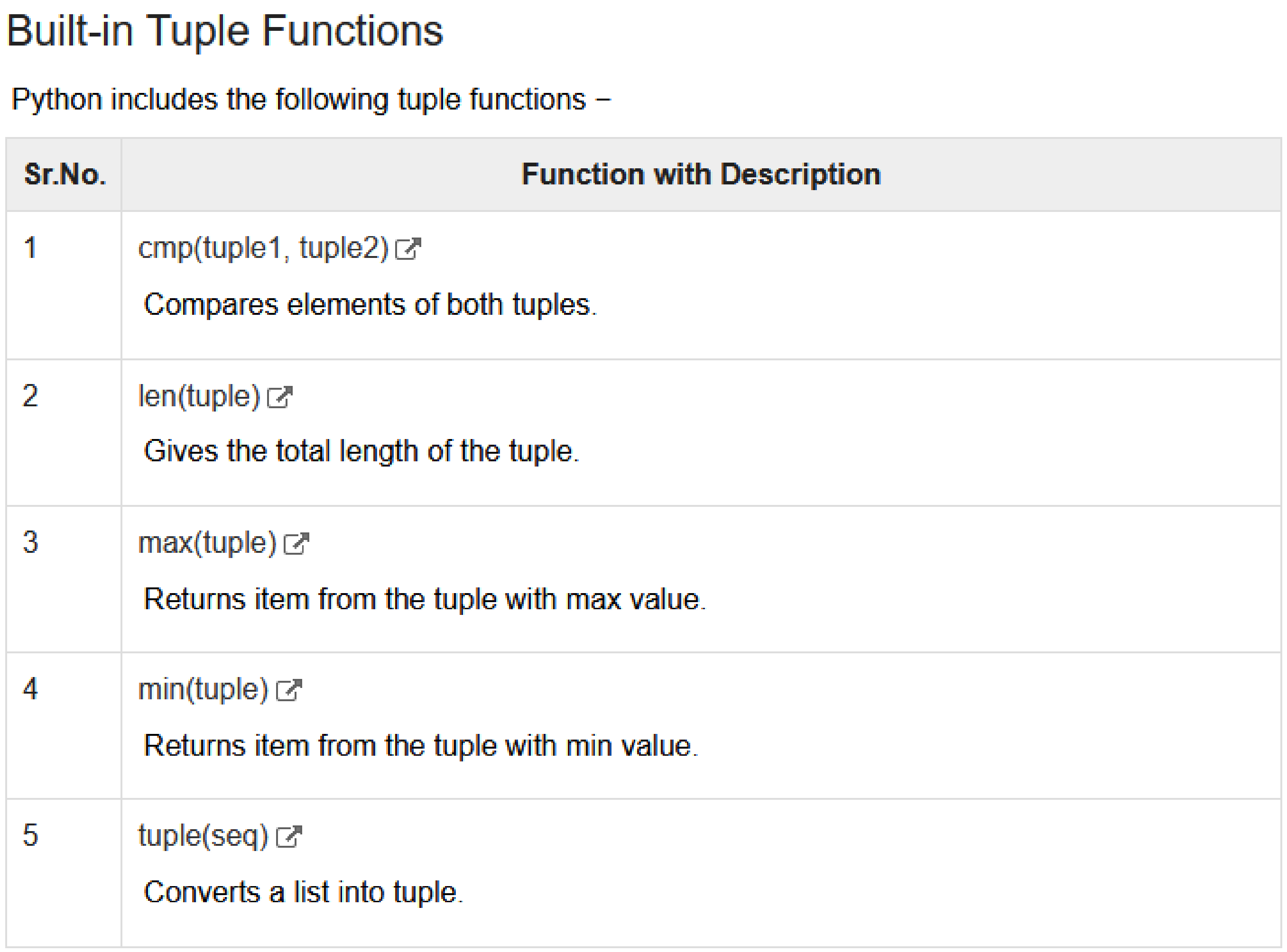
Tuple – Notice the trailing comma

```
b = ('a',2)
type(b)
Out[137]: tuple
```

Tuple – No trailing comma if there are 2 or more elements

Built in Tuple Functions

- Python has several built-in Tuple functions
- Tuples can be added using + operator
- Tuples can be repeated using * operator



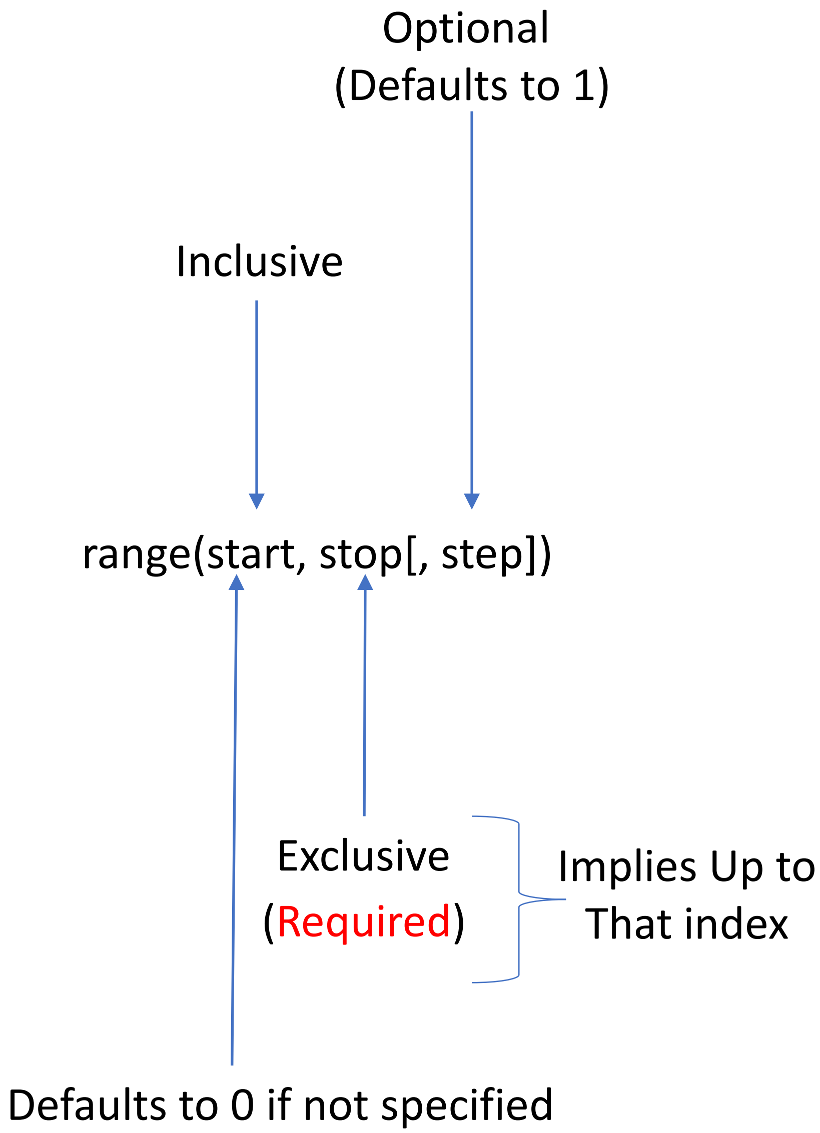
Built-in Tuple Functions

Python includes the following tuple functions −

Sr.No.	Function with Description
1	cmp(tuple1, tuple2) Compares elements of both tuples.
2	len(tuple) Gives the total length of the tuple.
3	max(tuple) Returns item from the tuple with max value.
4	min(tuple) Returns item from the tuple with min value.
5	tuple(seq) Converts a list into tuple.

Range

- Range is a special type of sequence
- Range only produces integer values
  - Cannot be used with string or float
- Range produces a sequence of values
  - A range object



Optional
(Defaults to 1)

Inclusive

range(start, stop[, step])

Exclusive
(Required)

Implies Up to
That index

Defaults to 0 if not specified

One can access the elements of a range object

Range

- Range object values are not visible
  - Can be converted to a list to see them

```
a = range(5) # create a range object
aa = list(a) # Convert to a list
aa # Write the list
Out[180]: [0, 1, 2, 3, 4]
```

Just
Stop
Value

```
b = range(1,5)
bb = list(b)
bb
Out[181]: [1, 2, 3, 4]
```

Start and Stop

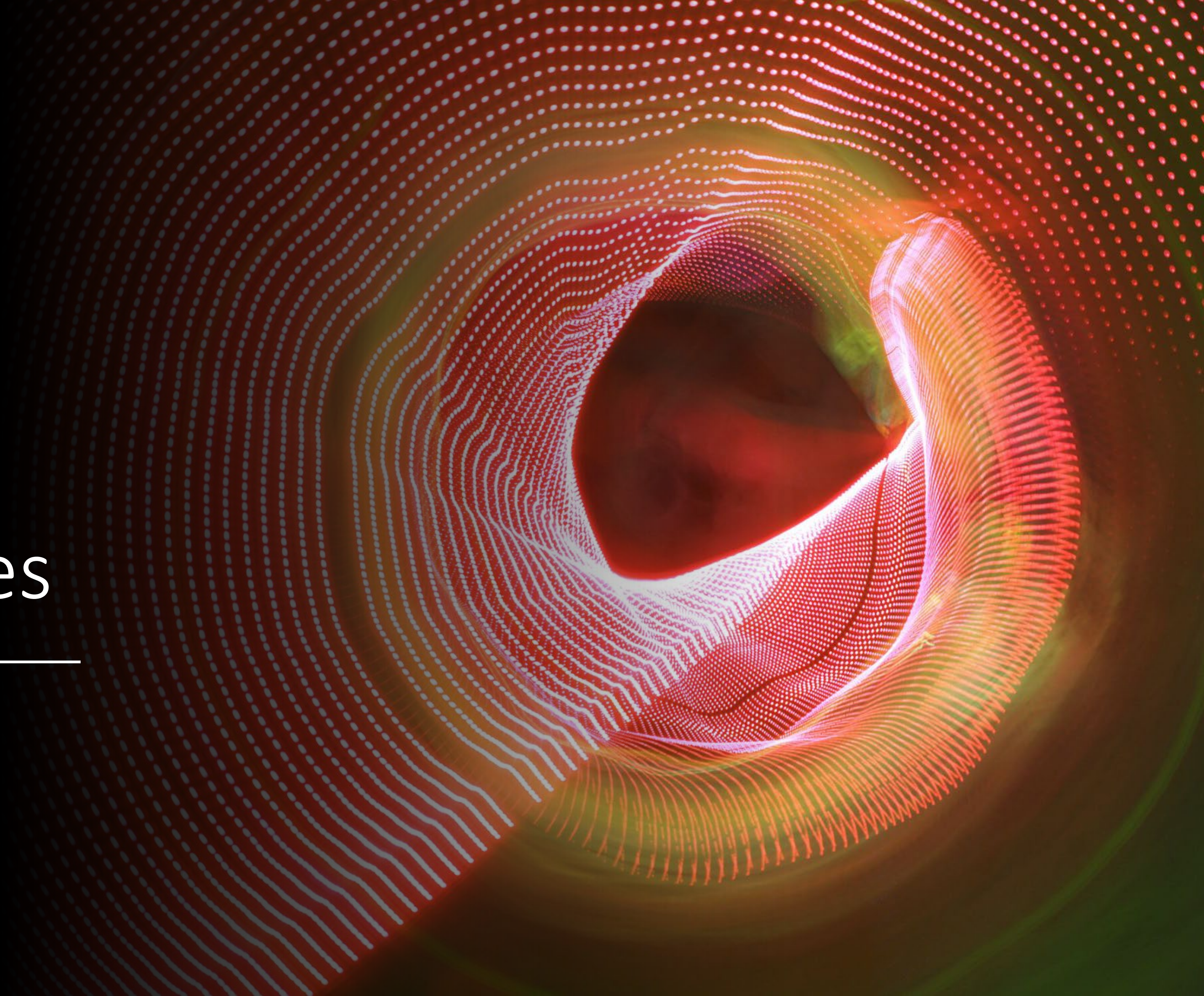
Start,Stop, Index

```
d = range(0,10,2)
dd = list(d)
dd
Out[183]: [0, 2, 4, 6, 8]
```

```
c = range(0,10,3)
cc = list(c)
cc
Out[182]: [0, 3, 6, 9]
```

Start,Stop, Index

Mapping Types



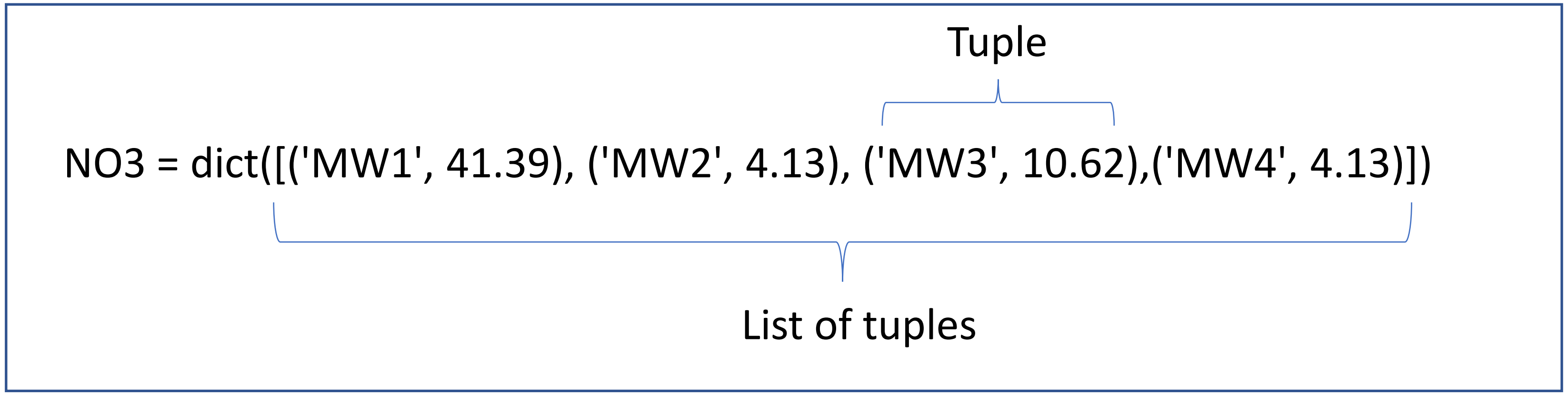
Dictionaries

- Dictionaries are associate type data structure
  - Characterized by a set of key:value pairs
  - A list of key:value pairs
- Keys are unique
  - Can be strings or numbers
  - Tuples containing strings and numbers can also be used
- Values need not be unique
  - Values can be replaced
  - Keys can be deleted
- Keys are used to extract values

A phone list is a classic example of a dictionary

Dictionaries

- Dictionaries can be built using the **dict()** command
  - **Each key-value pair is a tuple**
  - **A list of tuples is used to create a dictionary**



```
NO3 = dict([('MW1', 41.39), ('MW2', 4.13), ('MW3', 10.62),('MW4', 4.13)])
```

```
NO3['MW2'] # Extract value using the key
Out[188]: 4.13
```

```
[NO3[key] for key in ["MW1", "MW2"]]
Out[197]: [41.39, 4.13]
```

Curly Braces used to denote dictionary output

```
NO3
Out[189]: {'MW1': 41.39, 'MW2': 4.13, 'MW3': 10.62, 'MW4': 4.13}
```

Other Data Types

- Python has a few other data types
- Complex data type is used to define complex numbers
- Sets data type can be defined using set() function
  - Used to create sets
    - Unordered collection with no duplicate elements
  - Can perform union, intersection, difference and symmetric difference operations
- Python also supports iterators and generator types for iteration over containers