COMP7035

Python for Data Analytics and Artificial Intelligence

Data structures

Renjie Wan, Jun Qi

10/09/2024

香港浸會大學 HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE 計算機科學系

More about **for** loop

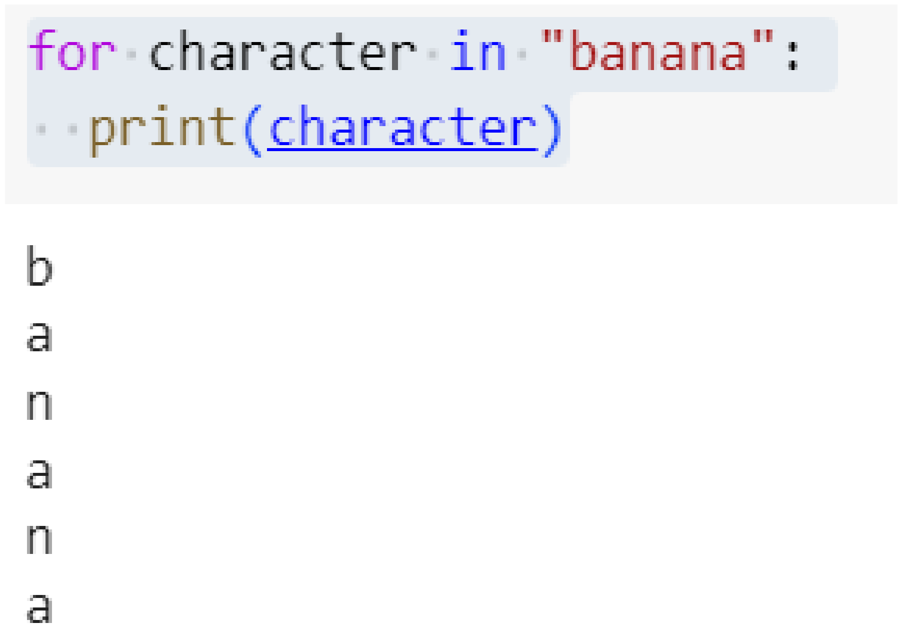
- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- Use string value as an example

A single element in this sequence

The sequence you want to iterate

```
for character in "banana":
  print(character)
```

You can process the element inside the for loop



```
for character in "banana":
  print(character)
```

```
b
a
n
a
n
a
```

More about range()

- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.
- Syntax: `range(start, stop, step)`

start	**Optional. An integer number specifying at which position to start. Default is 0**
stop	**Required. An integer number specifying at which position to stop (not included).**
step	**Optional. An integer number specifying the incrementation. Default is 1**

Question 1: Create a sequence of numbers from 3 to 5, and print each item in the sequence.

How to write the **range** function?

If you want to print 5, you need to write the stop number as 6, since it is not included!

More about range()

- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.
- Syntax: `range(start, stop, step)`

start	**Optional. An integer number specifying at which position to start. Default is 0**
stop	**Required. An integer number specifying at which position to stop (not included).**
step	**Optional. An integer number specifying the incrementation. Default is 1**

Question 1: Create a sequence of numbers from 3 to 5, and print each item in the sequence.

```
x = range(3, 6)
for n in x:
  print(n)
```

More about range()

- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.
- Syntax: `range(start, stop, step)`

start	**Optional. An integer number specifying at which position to start. Default is 0**
stop	**Required. An integer number specifying at which position to stop (not included).**
step	**Optional. An integer number specifying the incrementation. Default is 1**

Create a sequence of numbers from 3 to 19, but increment by 2 instead of 1:

```
x = range(3, 20, 2)
for n in x:
  print(n)
```

More about range()

- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.
- Syntax: `range(start, stop, step)`

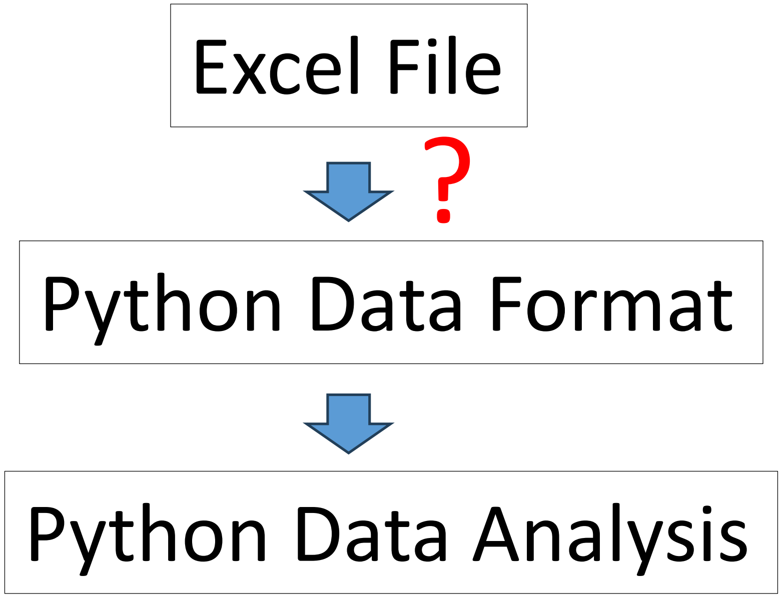
start	**Optional. An integer number specifying at which position to start. Default is 0**
stop	**Required. An integer number specifying at which position to stop (not included).**
step	**Optional. An integer number specifying the incrementation. Default is 1**

Create a sequence of numbers from 3 to 19, but increment by 2 instead of 1:

Some Exercises For You

A Scenario

- You have an Excel File with multiple values for employees’ salary.
- You hope to analyze those data via Python.
- Can we find a suitable format to store those data in Python and then conduct suitable analysis?



Data structures in Python

- Built-in Data Structures in Python
    - Lists
    - Tuples
    - Sets
    - Dictionary

Create a list

- An ordered group of items
  - Order means that their order has been predefined.
- You can create your list using the following way:

```
a = [1, 'This is a list', 'c', 1.5]
```

- Or you can also create a list using a **for** loop like:

```
x = [i for i in range(5)]
```

DEPARTMENT OF COMPUTER SCIENCE 計算機科學系

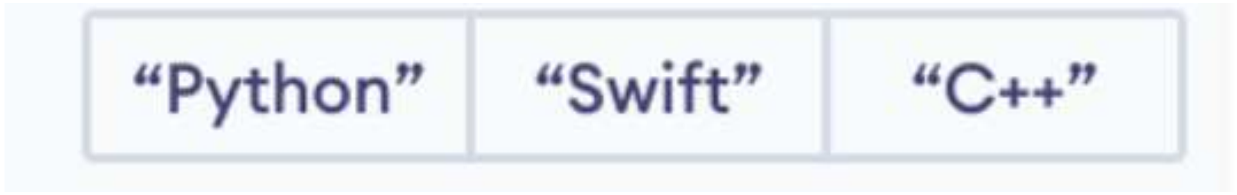
香港浸會大學 HONG KONG BAPTIST UNIVERSITY

Create a list

- You can create your list using the following way:

```
a = [1, 'This is a list', 'c', 1.5]
```

- Please create a list with the following elements:



"Python"	"Swift"	"C++"

DEPARTMENT OF COMPUTER SCIENCE 計算機科學系

香港浸會大學 HONG KONG BAPTIST UNIVERSITY

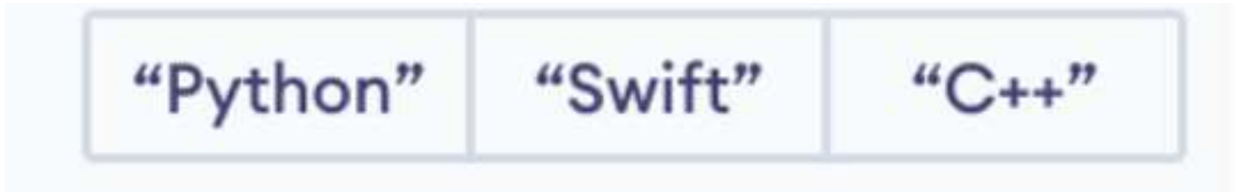
Create a list

- You can create your list using the following way:

E1

```
a = [1, 'This is a list', 'c', 1.5]
```

- Please create a list with the following elements:



“Python”	“Swift”	“C++”

```
c = [“Python”, “Swift”, “C++”]
```

Create a list

- A list can
  - store elements of different types (integer, float, string, etc.)
  - store duplicate elements

```
# list with elements of different data types
list1 = [1, "Hello", 3.4]
# list with duplicate elements
list1 = [1, "Hello", 3.4, "Hello", 1] # empty list
list3 = []
```

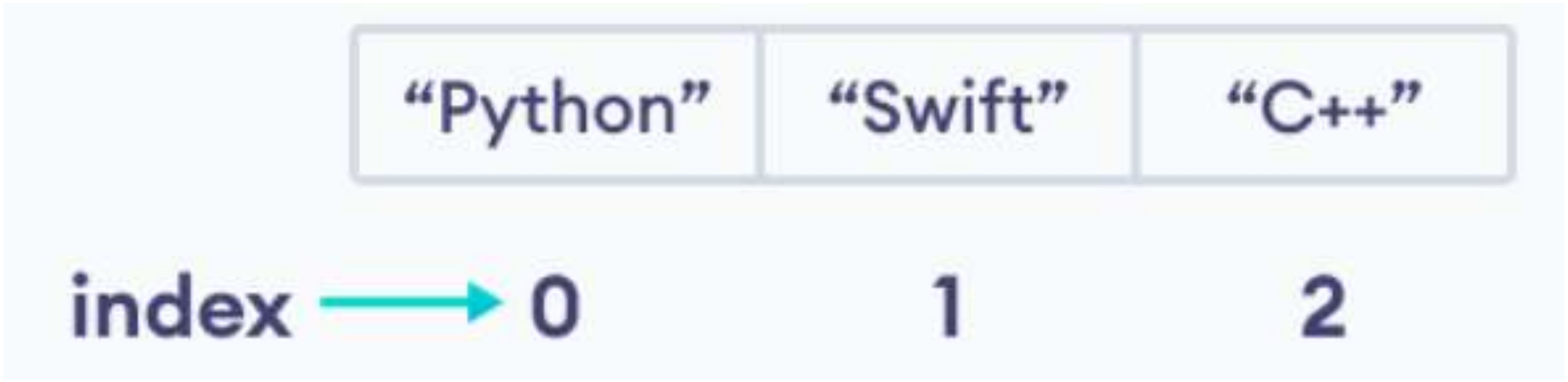
Create a list

- The list() constructor returns a list in Python.

```
text = 'Python'
# convert string to list
text_list = list(text)
print(text_list)
# check type of text_list
print(type(text_list))
# Output: ['P', 'y', 't', 'h', 'o', 'n'] # <class 'list'>
```

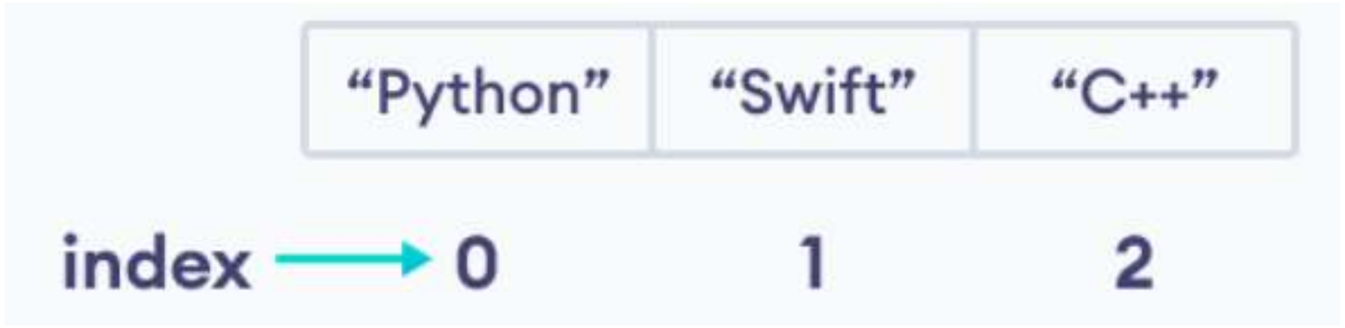
Python Indexing and Slicing

- Indexing is the process of accessing an element in a sequence using its position in the sequence (its index).
- In Python, indexing starts from 0, which means the first element in a sequence is at position 0, the second element is at position 1, and so on.
- To access an element in a sequence, you can use square brackets [] with the index of the element you want to access.



Python Indexing and Slicing

- Indexing is the process of accessing an element in a sequence using its position in the sequence (its index).
- In Python, indexing starts from 0, which means the first element in a sequence is at position 0, the second element is at position 1, and so on.
- To access an element in a sequence, you can use square brackets [] with the index of the element you want to access.

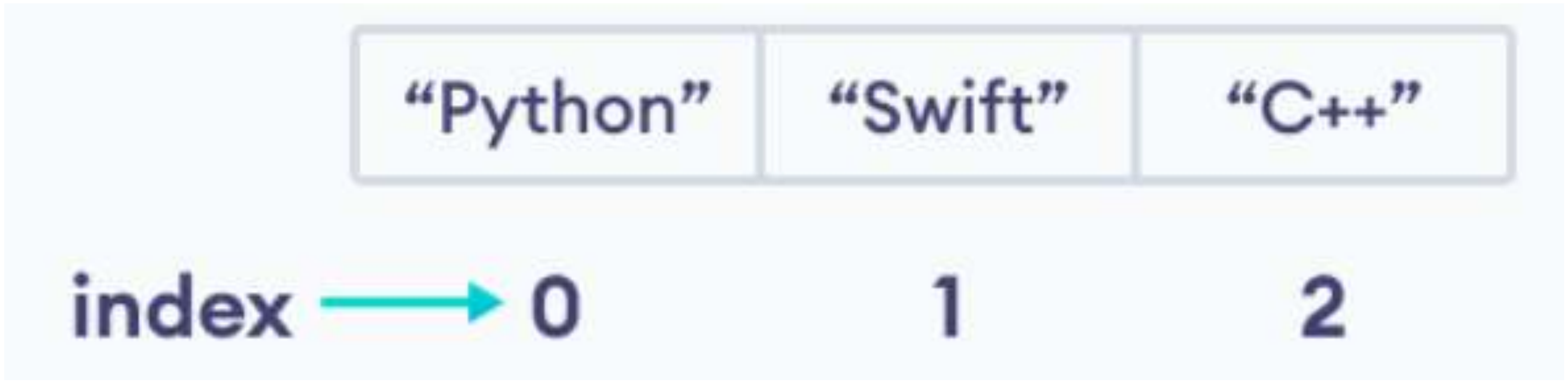


Access the list elements like: list_name[index]

Python Indexing and Slicing

```
c = ["Python", "Swift", "C++"]
```

c[1]   c[0]   c[3]   c[2]

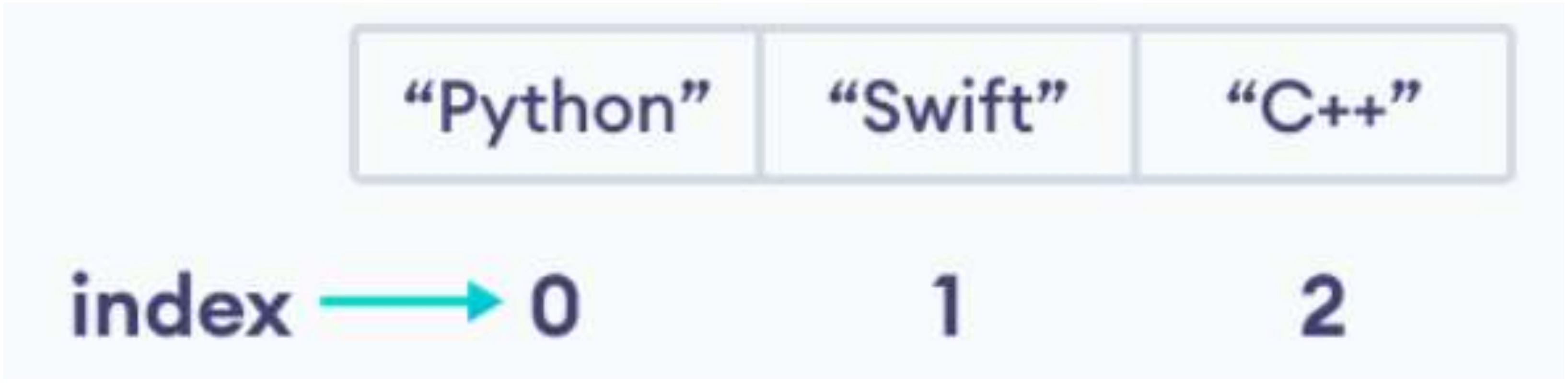


Python Indexing and Slicing

```
c = [“Python”, “Swift”, “C++”]
```

c[1]　c[0]　c[3]　c[2]

```
“Swift”  “Python” Error  “C++”
```



Python Indexing and Slicing

```
c = [“Python”, “Swift”, “C++”]
```

c[1]	c[0]	c[3]	c[2]
“Swift”	“Python”	Error	“C++”

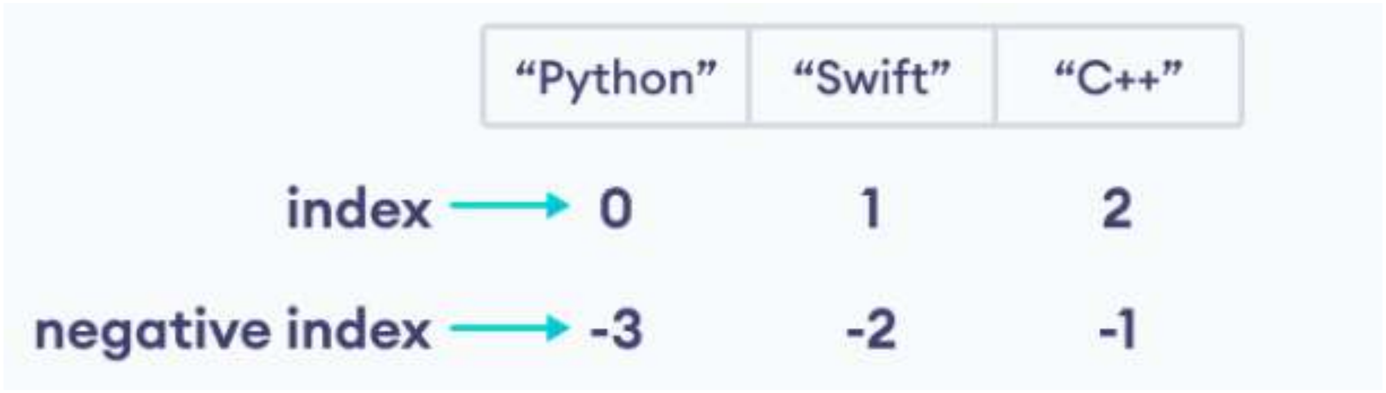
The list index always starts with 0. Hence, the first element of a list is present at index 0, not 1.

Python Indexing and Slicing

- Negative Indexing
- Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

c[1]   c[0]   c[3]   c[2]

Please use negative index to achieve the same functions!

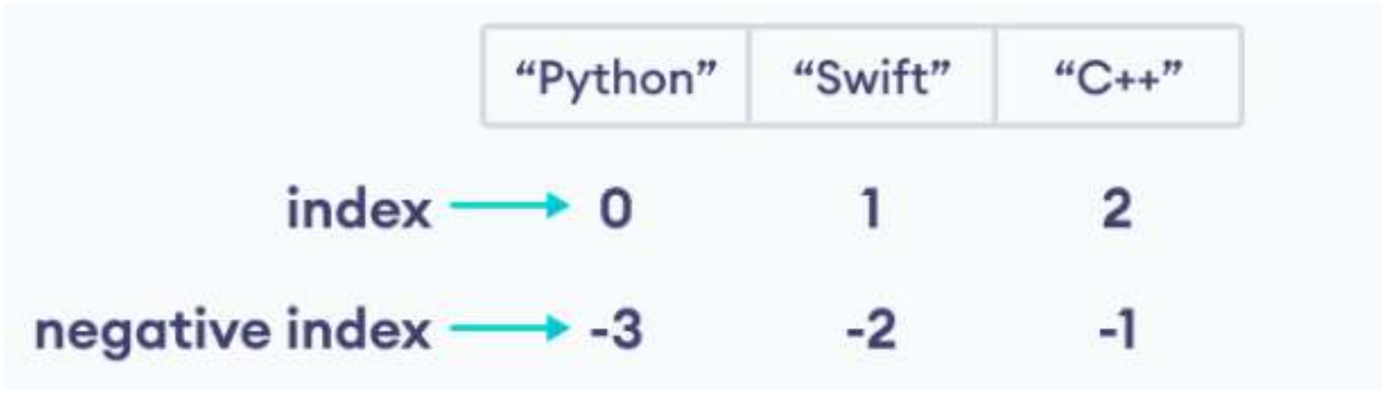


Python Indexing and Slicing

- Negative Indexing
- Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

c[1]   c[0]   c[3]   c[2]

Please use negative index to achieve the same functions!



Python Indexing and Slicing

- Slicing is the process of accessing a sub-sequence of a sequence by specifying a starting and ending index using the slicing operator **:**

```
sequence[start_index:end_index]
```

```
my_list = ['apple', 'banana', 'cherry', 'date’]
print(my_list[1:3])
```

Please try above codes.

Python Indexing and Slicing

- Slicing is the process of accessing a sub-sequence of a sequence by specifying a starting and ending index using the slicing operator **:**

```
sequence[start_index:end_index]
```

```
my_list = ['apple', 'banana', 'cherry', 'date’]
print(my_list[1:3])
```

```
output: ['banana', 'cherry']
```

Note: When we slice lists, the **start index is inclusive**, but the **end index is exclusive**.

Python Indexing and Slicing

Note: When we slice lists, the **start index is inclusive**, but the **end index is exclusive**.

```
my_list = ['apple', 'banana', 'cherry', 'date’]
```

Please try the following codes:

```
my_list[0:3]
my_list[1:4]
```

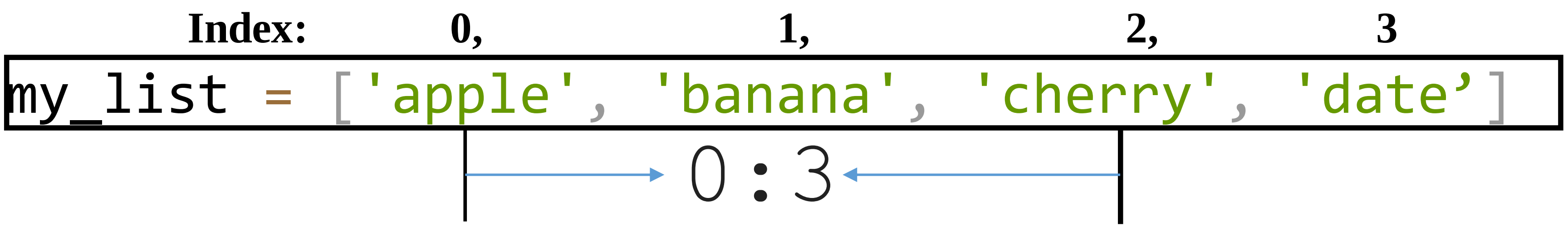
Python Indexing and Slicing

Note: When we slice lists, the **start index is inclusive**, but the **end index is exclusive**.

```
my_list = ['apple', 'banana', 'cherry', 'date’]
```

Please try the following codes:

```
my_list[0:3]->['apple', 'banana', 'cherry’]
my_list[1:4]->['banana', 'cherry', 'date']
```



Index: **0,** **1,** **2,** **3**

```
my_list = ['apple', 'banana', 'cherry', 'date’]
```

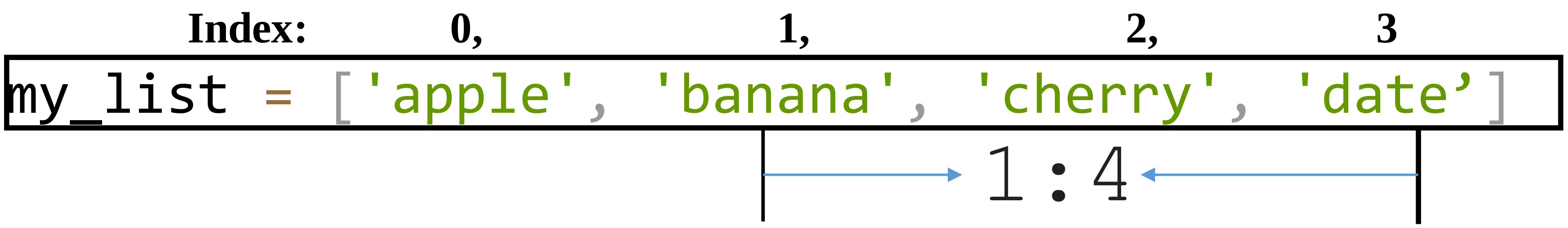
Python Indexing and Slicing

Note: When we slice lists, the **start index is inclusive**, but the **end index is exclusive**.

```
my_list = ['apple', 'banana', 'cherry', 'date’]
```

Please try the following codes:

```
my_list[0:3]->['apple', 'banana', 'cherry’]
my_list[1:4]->['banana', 'cherry', 'date']
```



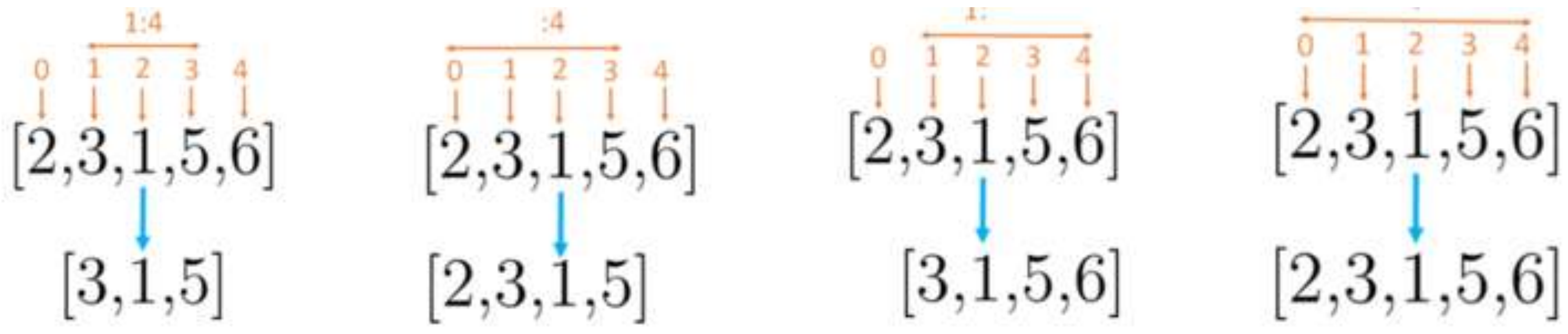
Python Indexing and Slicing

- You can also omit either the start_index or the end_index in a slice to get all the elements from the beginning or end of the sequence.
- By leaving out the start index, the range will start at the first element.
- By leaving out the end index, the range will go to the end.
- my_list[:] returns all list items

```
my_list = ['apple', 'banana', 'cherry', 'date’]
print(my_list[:2]) # output: ['apple', 'banana’]
print(my_list[2:]) # output: ['cherry', 'date']
```

Let us do a summary

- For slice, we provide a start and an end number separated by a semicolon (:). The range then starts at the start number and one before the end number.
- When you want to get the whole elements from the start until the element with index 3, I could write: print(arr[0:4]).
- To get from the first index all the way to the end of the array, I can write it without providing a slicing end



Some exercises for you in the lab materials.

Properties of List

- Lists can be sliced
    - We can use Python Slicing
- Lists can be replicated by using multiplication operator (*)

```
myList = ['hello', 'world']
print(myList * 2)
print(myList * 3)
```

Properties of List

- Lists can be sliced
    - We can use Python Slicing
- Lists can be replicated by using multiplication operator (*)

```
myList = ['hello', 'world']
print(myList * 2)
print(myList * 3)
```

```
['hello', 'world', 'hello', 'world’]
['hello', 'world', 'hello', 'world', 'hello', 'world']
```

Properties of List

- Lists can be sliced
  - We can use Python Slicing
- Lists can be replicated by using multiplication operator (*)
- Lists can be combined by using addition operator (+)

```
myList = [5, 2.3, 'hello']
slicedlist = myList[0:2]
mySecondList = ['a', '3']
concatList = myList + mySecondList
# [5, 2.3, 'hello', 'a', '3'] the two lists are added here
```

Properties of List

- Lists can be sliced
  - We can use Python Slicing
- Lists can be replicated by using multiplication operator (*)
- Lists can be combined by using addition operator (+)

```
myList = [5, 2.3, 'hello']
slicedlist = myList[0:2]
mySecondList = ['a', '3']
concatList = myList + mySecondList
```

Properties of List

- Lists Are Ordered
- Lists that have the same elements in a different order are not the same:

```
a = ['foo', 'bar', 'baz', 'qux']
b = ['baz', 'qux', 'bar', 'foo']

print(a == b)
print(a is b)
print(a != b)
print(a is not b)
```

Properties of List

- Lists Can Contain Arbitrary Objects
- A list can contain any assortment of objects. The elements of a list can all be the same type.

```
a = [21.42, 'foobar', 3, 4, 'bark', False, 3.14159]
```

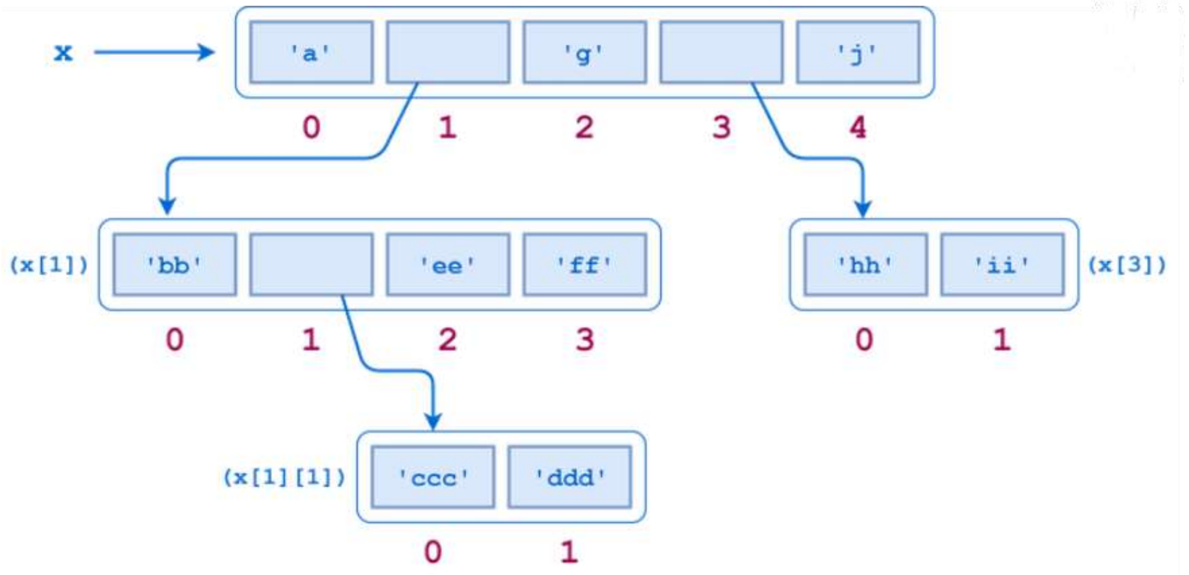
- A list can contain any number of objects, from zero to as many as your computer’s memory will allow:

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,
65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96,
97, 98, 99, 100]
```

Properties of List

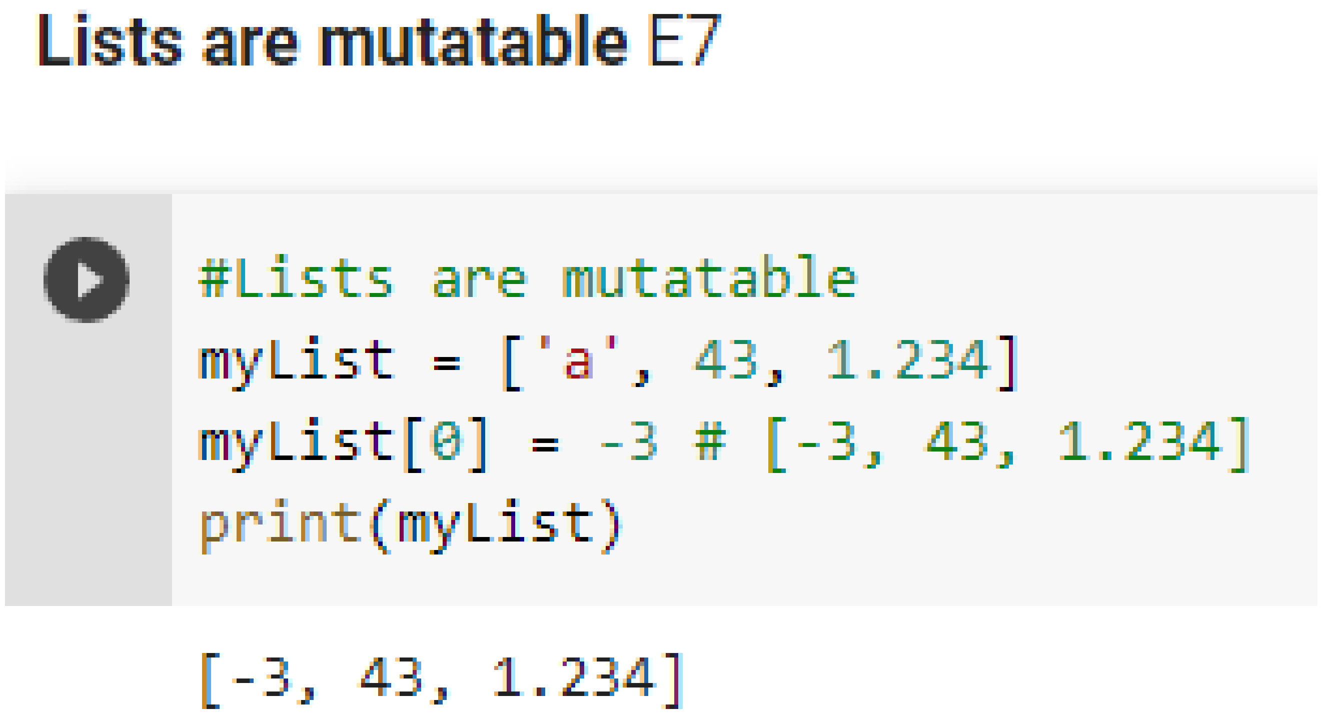
- Lists Can Be Nested
- A list can contain sublists, which in turn can contain sublists themselves, and so on to arbitrary depth.

```python
x = ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']
```



Lists are mutable

- Lists are mutable, this means that individual elements can be changed.



Lists are mutatable E7

```
#Lists are mutatable
myList = ['a', 43, 1.234]
myList[0] = -3 # [-3, 43, 1.234]
print(myList)
```

```
[-3, 43, 1.234]
```

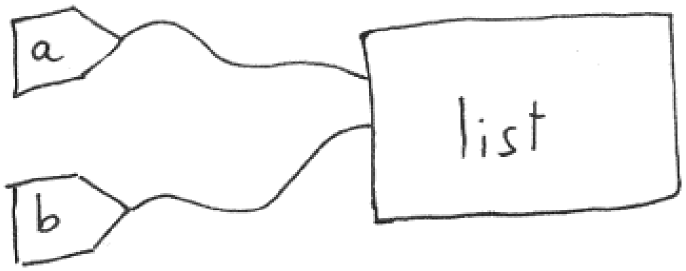
Copying a list

- How to copy a list?

```
a = ['a', 'b', 'c']
b = a # let’s copy list1
print(b)
b[1] = 1 # now we want to change an element
print(b) # [’a’, 1, ’c’]
print(a) # [’a’, 1, ’c’]
```

What just happened?

- Variables in Python really are tags:



So b = a means: b is same tag as a.

Image from http://henry.precheur.org/python/copy_list.html

Copying a list

- Instead: we want:

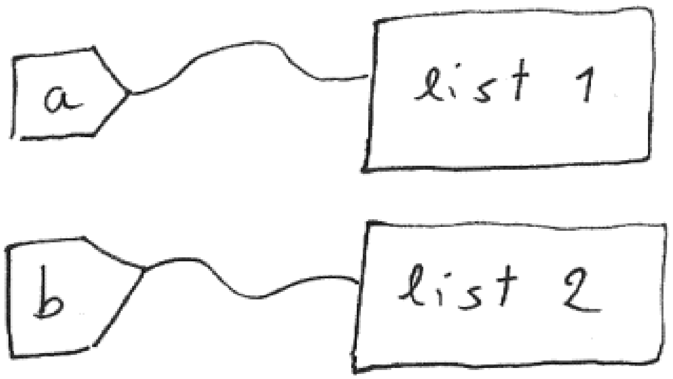
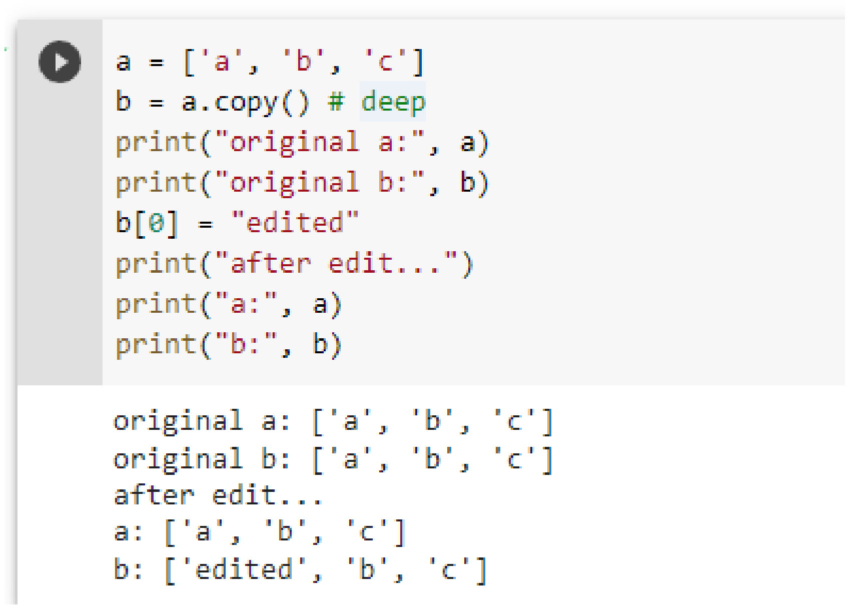


Figure: b = list(a) or b = a[:]

Image from http://henry.precheur.org/python/copy_list.html

Copying a list

Try some different ways like List.copy() E9



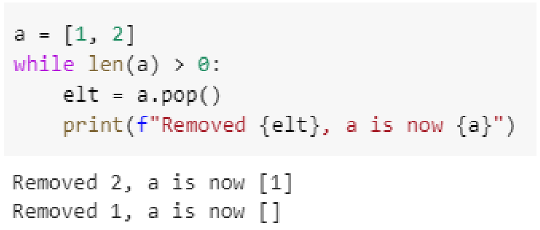
```
a = ['a', 'b', 'c']
b = a.copy() # deep
print("original a:", a)
print("original b:", b)
b[0] = "edited"
print("after edit...")
print("a:", a)
print("b:", b)
```

```
original a: ['a', 'b', 'c']
original b: ['a', 'b', 'c']
after edit...
a: ['a', 'b', 'c']
b: ['edited', 'b', 'c']
```

Copying a list

- Make a copy of a list with the copy() method:

```
a = ["apple", "banana", "cherry"]
b = a.copy()
print("original a:", a)
print("original b:", b)
b[0] = "edited"
print("after edit...")
print("a:", a)
print("b:", b)
```

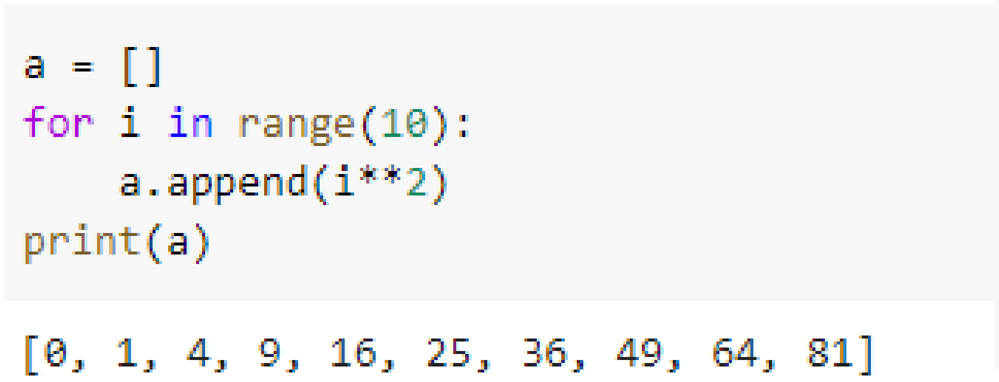


```python
a = [1, 2]
while len(a) > 0:
    elt = a.pop()
    print(f"Removed {elt}, a is now {a}")
```

```
Removed 2, a is now [1]
Removed 1, a is now []
```

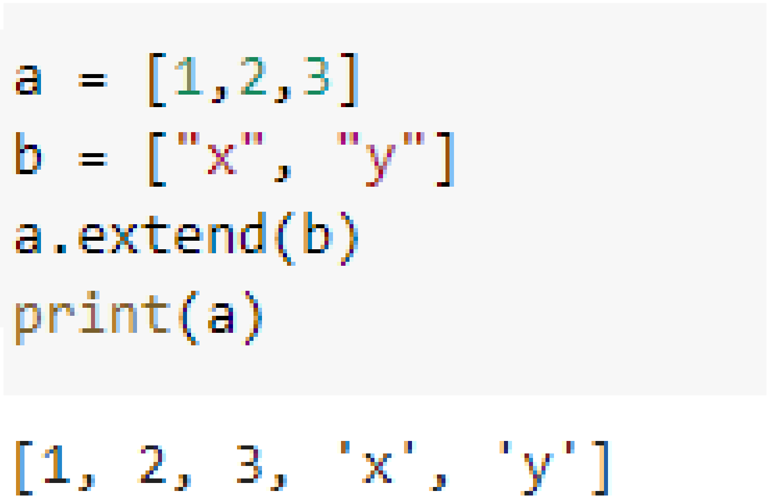
Methods of Lists

- List.append(x)
  - adds an item to the end of the list
- List.extend(L)
  - Extend the list by appending all in the given list L



```python
a = []
for i in range(10):
    a.append(i**2)
print(a)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



```python
a = [1,2,3]
b = ["x", "y"]
a.extend(b)
print(a)
```

```
[1, 2, 3, 'x', 'y']
```

Methods of Lists

- List.append(x)
    - adds an item to the end of the list

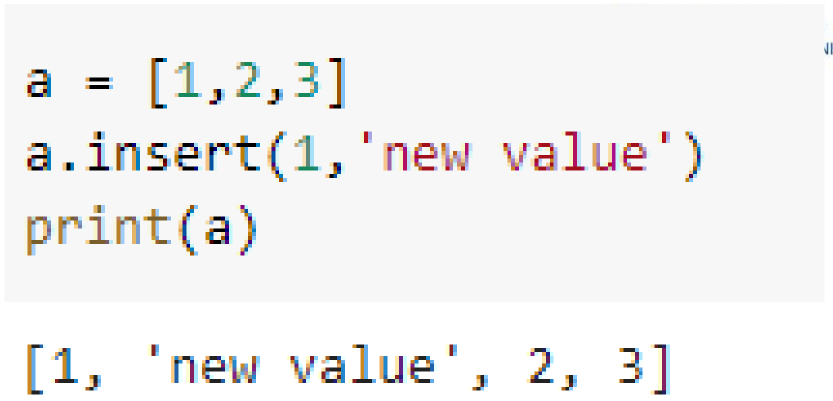
```
fruits = ['apple', 'banana', 'cherry']
fruits.append("orange")
```

Methods of Lists

- List.extend(L)
  - Extend the list by appending all in the given list L

```
fruits = ['apple', 'banana', 'cherry']

cars = ['Ford', 'BMW', 'Volvo']

fruits.extend(cars)
```

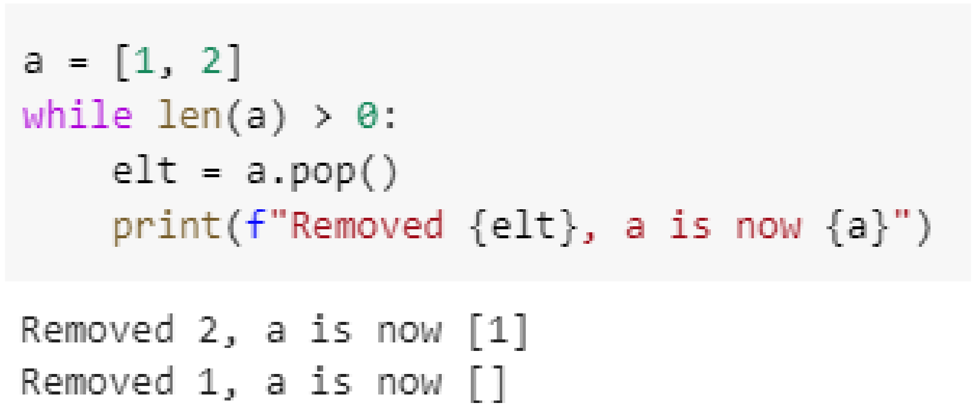
Methods of Lists



```
a = [1,2,3]
a.insert(1,'new value')
print(a)
```

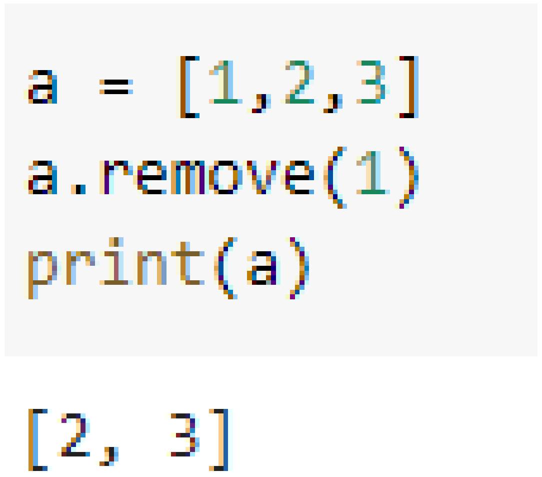
```
[1, 'new value', 2, 3]
```

- List.pop(index)
  - Remove the item in the index position and return the deleted value
- List.insert(i, x)
  - Inserts an item at index i
- List.remove(x)
  - Removes the first item from the list whose value is x



```
a = [1, 2]
while len(a) > 0:
    elt = a.pop()
    print(f"Removed {elt}, a is now {a}")
```

```
Removed 2, a is now [1]
Removed 1, a is now []
```



```
a = [1,2,3]
a.remove(1)
print(a)
```

```
[2, 3]
```

Methods of Lists

- List.pop(index)
  - Remove the item in the index position and return the deleted value

```
fruits = ['apple', 'banana', 'cherry']

fruits.pop(1)
```

Methods of Lists

- List.insert(i, x)
  - Inserts an item at index i

```
fruits = ['apple', 'banana', 'cherry']

fruits.insert(1, "orange")
```

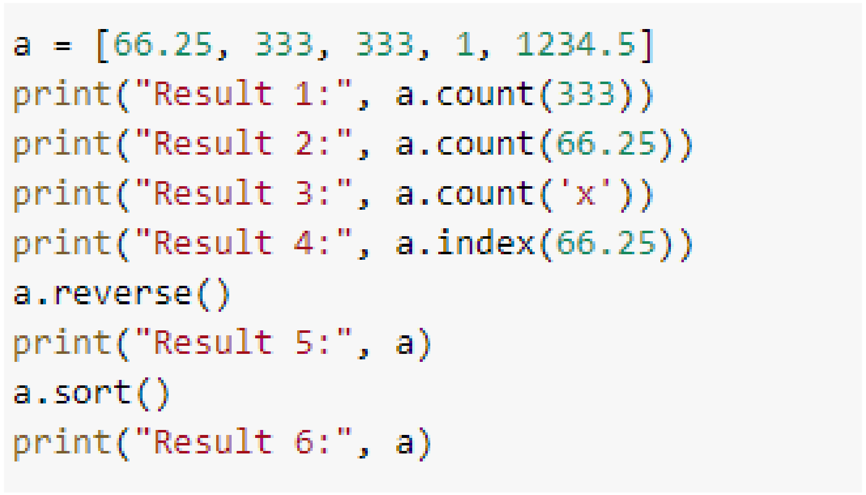
Methods of Lists

- List.remove(x)
  - Removes the first item from the list whose value is x

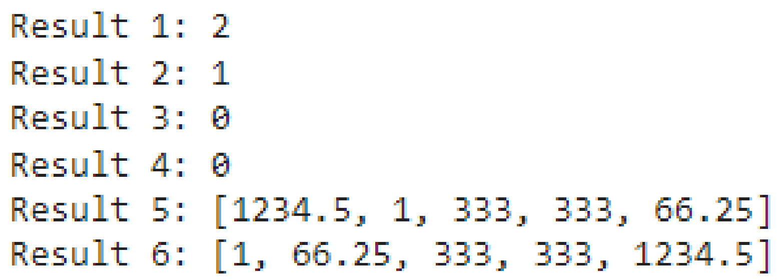
```
fruits = ['apple', 'banana', 'cherry']

fruits.remove("banana")
```

Examples of other methods

- a.index()
  - Returns the first index where the given value appears
- a.reverse()
  - Reverses order of list
- a.sort()
  - Sorting the list in ascending order



```python
a = [66.25, 333, 333, 1, 1234.5]
print("Result 1:", a.count(333))
print("Result 2:", a.count(66.25))
print("Result 3:", a.count('x'))
print("Result 4:", a.index(66.25))
a.reverse()
print("Result 5:", a)
a.sort()
print("Result 6:", a)
```



```
Result 1: 2
Result 2: 1
Result 3: 0
Result 4: 0
Result 5: [1234.5, 1, 333, 333, 66.25]
Result 6: [1, 66.25, 333, 333, 1234.5]
```

Methods of Lists

- List.reverse()
    - Reverses order of list

```
fruits = ['apple', 'banana', 'cherry']

fruits.reverse()
```

List comprehensions

- Python's list comprehensions let you create lists in a way that is reminiscent of set notation

$$S = \{x | 0 \leq \mathrm{x} \leq 20, \quad \mathrm{x}\ mod\ 3 = 0\}$$

- We need all numbers among 0 to 20 and they should be able to be divided by 3.

For x from 0 to 20, add the number to a list if it is can be divided by 3

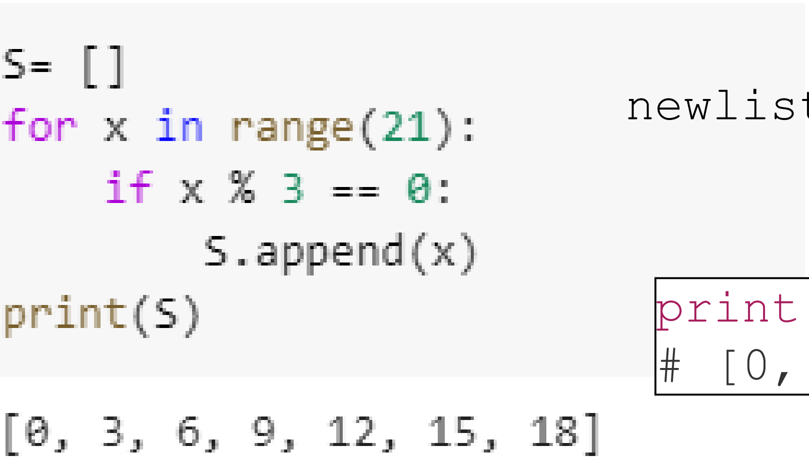
List comprehensions

- Python's list comprehensions let you create lists in a way that is reminiscent of set notation

$$S = \{x | 0 \le x \le 20, \quad x \; mod \; 3 = 0\}$$

- We need all numbers among 0 to 20 and they should be able to be divided by 3.

For x from 0 to 20, add the number to a list if it is can be divided by 3



```
S= []
for x in range(21):
    if x % 3 == 0:
        S.append(x)
print(S)
```

```
[0, 3, 6, 9, 12, 15, 18]
```

```
newlist = [x for x in range(21) if x%3 == 0]
```

```
print [i**2 for i in range(5)]
# [0, 1, 4, 9, 16]
```

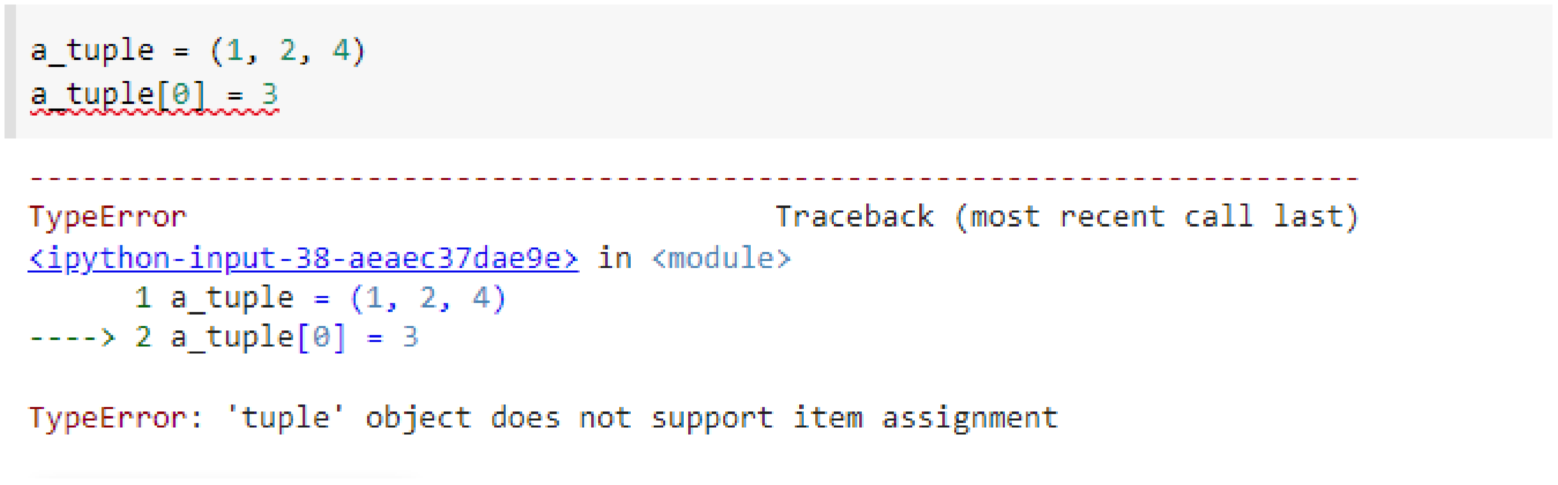
One More Question

- What is the role of this code?

```python
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

Other collection: Tuples

- List is ordered, indexed, and mutable
- Tuple: Ordered, indexed, but immutable
    - A number of values separated by commas
    - Immutable
        - Cannot assign values to individual items of a tuple
        - However, tuples can contain mutable objects such as lists



```
a_tuple = (1, 2, 4)
a_tuple[0] = 3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-38-aeaec37dae9e> in <module>
      1 a_tuple = (1, 2, 4)
----> 2 a_tuple[0] = 3

TypeError: 'tuple' object does not support item assignment
```

Tuples

- Tuples are written with round brackets.
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.
- Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

```
thistuple = ["apple", "banana", "cherry"]
thistuple = ("apple", "banana", "cherry")
```

This is a list

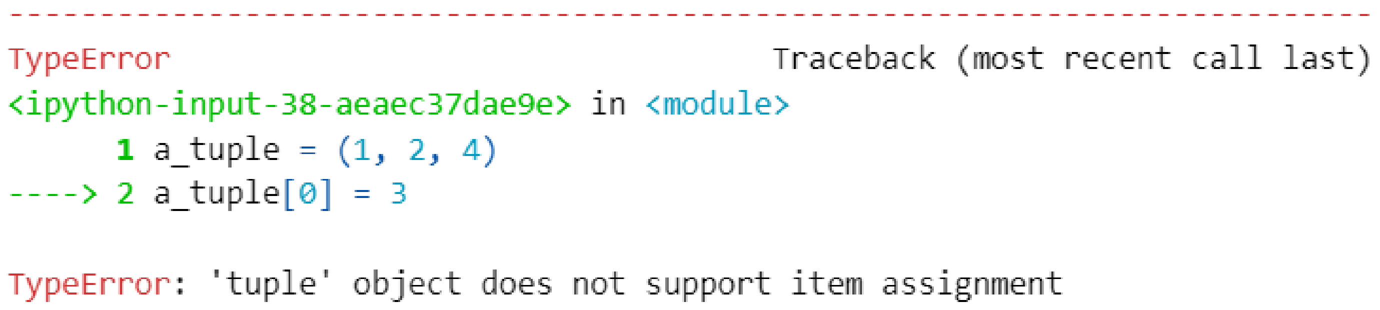
This is a tuple

Tuples

- Tuples are written with round brackets.
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.
- Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

```
a_tuple = (1, 2, 4)
a_tuple[0] = 3
```

Items in tuples cannot be changed



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-38-aeaec37dae9e> in <module>
      1 a_tuple = (1, 2, 4)
----> 2 a_tuple[0] = 3

TypeError: 'tuple' object does not support item assignment
```

DEPARTMENT OF COMPUTER SCIENCE 計算機科學系

香港浸會大學 HONG KONG BAPTIST UNIVERSITY

Creation of Tuples

- A tuple is created by placing all the items (elements) inside parentheses (), separated by commas.

```python
# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)
```

```python
# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

Create a Tuple With one Element

- In Python, creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough.

```
var1 = ("hello")
print(type(var1))

# Creating a tuple having one element
var2 = ("hello",)
print(type(var2))
```

What is the difference?

Create a Tuple With one Element

- In Python, creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough.
- ("hello") is a string so type() returns str as class of var1 i.e. <class 'str'>
- ("hello",) and "hello", both are tuples so type() returns tuple as class of var1 i.e. <class 'tuple'>

```
var1 = ("hello")
print(type(var1))  # <class 'str'>

# Creating a tuple having one
element
var2 = ("hello",)
print(type(var2))  # <class 'tuple'>
```

Create a Tuple With one Element

- In Python, creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough.
- ("hello") is a string so type() returns str as class of var1 i.e. <class 'str'>
- ("hello",) and "hello", both are tuples so type() returns tuple as class of var1 i.e. <class 'tuple'>
- We will need a trailing comma to indicate that it is a tuple,

```
var1 = ("hello")
print(type(var1))  # <class 'str'>

# Creating a tuple having one
element
var2 = ("hello",)
print(type(var2))  # <class 'tuple'>
```

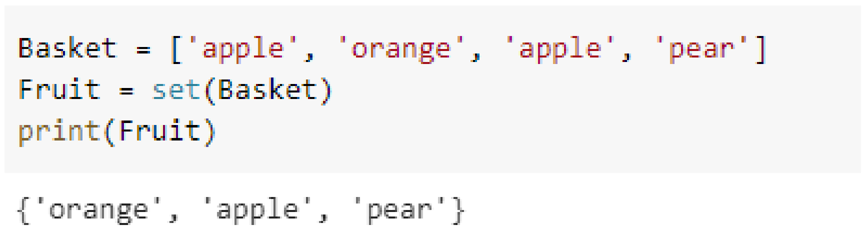
Access Python Tuple Elements

- Like a list, each element of a tuple is represented by index numbers (0, 1, ...) where the first element is at index 0.
- Indexing rule is same to list

```
# accessing tuple elements using indexing
letters = ("p", "r", "o", "g", "r", "a", "m", "i", "z")

print(letters[0])   # prints "p"
print(letters[5])   # prints "a"
```

Other collection: Sets

- An unordered collection with no duplicate elements
- Unordered, unindexed, mutable, and doesn't allow for duplicate elements



```python
Basket = ['apple', 'orange', 'apple', 'pear']
Fruit = set(Basket)
print(Fruit)
```

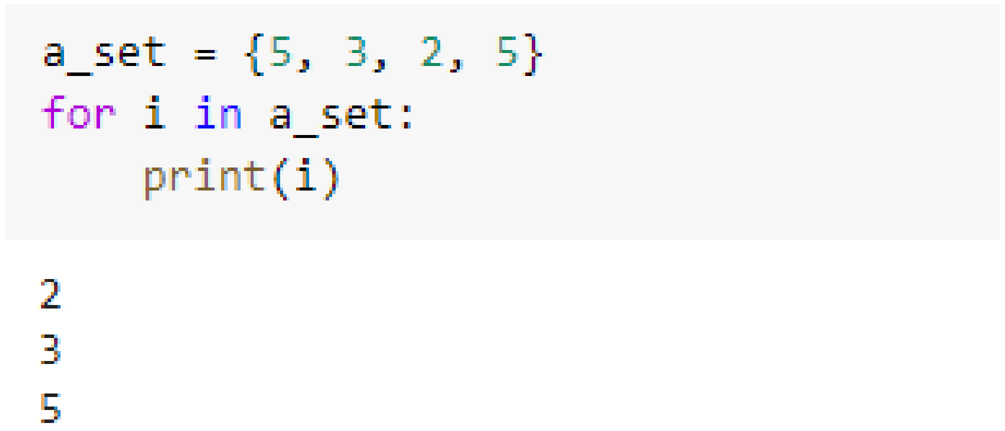
```
{'orange', 'apple', 'pear'}
```

- Only one ‘apple’ is left

Other collection: Sets

- An unordered collection with no duplicate elements
- Unordered, unindexed, mutable, and doesn't allow for duplicate elements

You can also define a set as



```python
a_set = {5, 3, 2, 5}
for i in a_set:
    print(i)
```

```
2
3
5
```

But a_set only contains 5, 3, 2， since no duplicate is allowed

Sets

- An unordered collection with no duplicate elements
- Doesn't allow for duplicate elements

```
a_set = {5, 3, 2, 5}
for i in a_set:
    print(i)
```

Please try above codes yourself

Three ways to create Sets

- First way: using the set() function on an iterable object

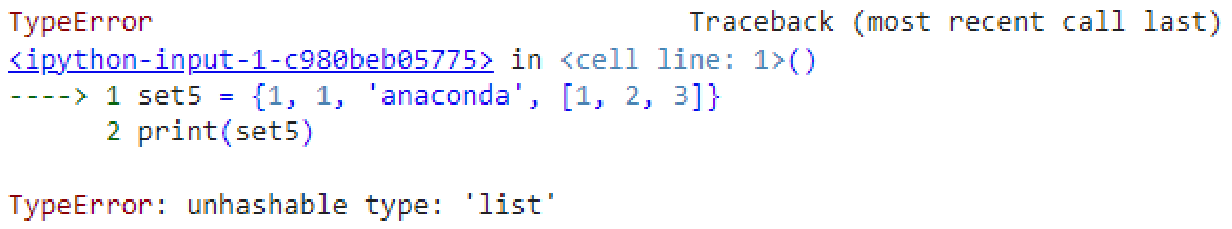
```
set1 = set([1, 1, 1, 2, 2, 3])          # from a list
set2 = set(('a', 'a', 'b', 'b', 'c'))   # from a tuple
set3 = set('anaconda')                  # from a string
```

- Second way: using curly braces

```
set4 = {1, 1, 'anaconda', 'anaconda', 8.6, (1, 2, 3), None}
```

- Incorrect way: trying to create a set with mutable items (a list and a set)

```
set5 = {1, 1, 'anaconda', 'anaconda', 8.6, [1, 2, 3], {1, 2, 3}}
print('Set5:', set5)
```

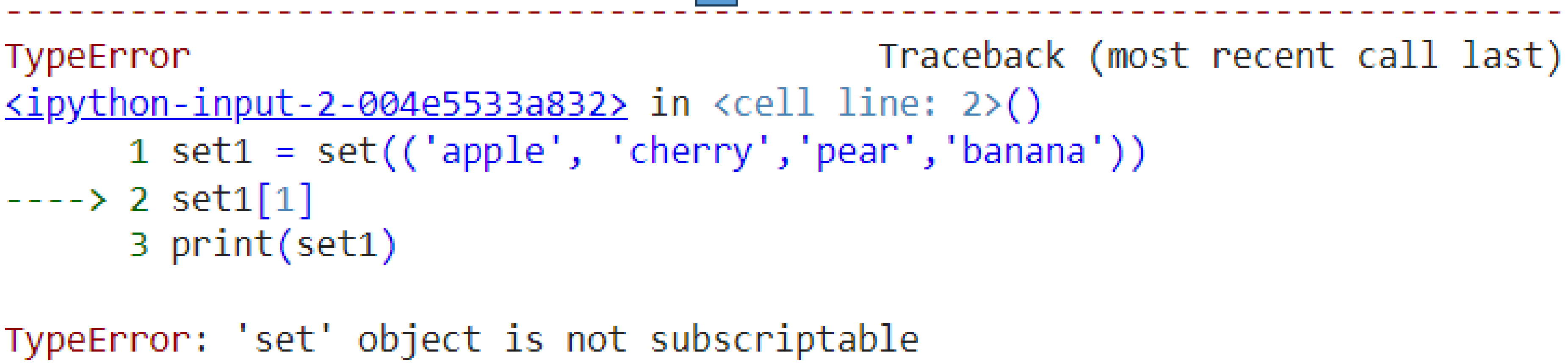


```
TypeError                                 Traceback (most recent call last)
<ipython-input-1-c980beb05775> in <cell line: 1>()
----> 1 set5 = {1, 1, 'anaconda', [1, 2, 3]}
      2 print(set5)

TypeError: unhashable type: 'list'
```

Python Set is unindexed

- You cannot access the items of a set

```
set1 = set(('apple', 'cherry','pear','banana'))
set1[1]
```



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-004e5533a832> in <cell line: 2>()
      1 set1 = set(('apple', 'cherry','pear','banana'))
----> 2 set1[1]
      3 print(set1)

TypeError: 'set' object is not subscriptable
```

Try more exercises

Dictionaries of Python

- Dictionaries are used to store data values in **key:value** pairs.

Employees' type	No. in this type
Male	**35**
Female	46
Senior People	10

```
thisdict = {
  “Male": 35,
  “Female": 46,
  “Senior People": 10
}
print(thisdict)
```

Dictionaries of Python

- Dictionaries are used to store data values in **key:value** pairs.
- Dictionary items can be referred to by using the key name.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

Dictionaries of Python

- Dictionaries are used to store data values in **key:value** pairs.
- Dictionary items can be referred to by using the key name.
- Duplicates are strictly not allowed: Duplicate values will overwrite existing values.
-

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```

Creation of Dictionaries

- We can use dict() constructor to build a dictionary.

```
thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)
```

```
{'name': 'John', 'age': '36', 'country': 'Norway'}
```