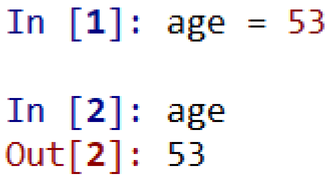
Lesson 4 – Data structures

Introduction to Data Structures

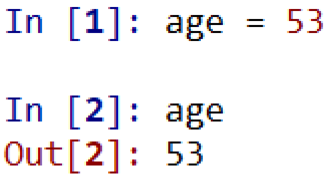
- Programs created till now have used objects of diferent type p.e `int`, `float`.
- The types `int` and `float` are scalar. This means that these objects do not have an accessible internal structure.



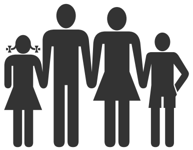
```
In [1]: age = 53

In [2]: age
Out[2]: 53
```

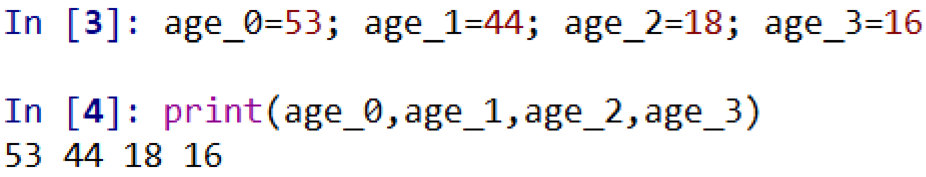
Introduction to Data Structures

- Motivation

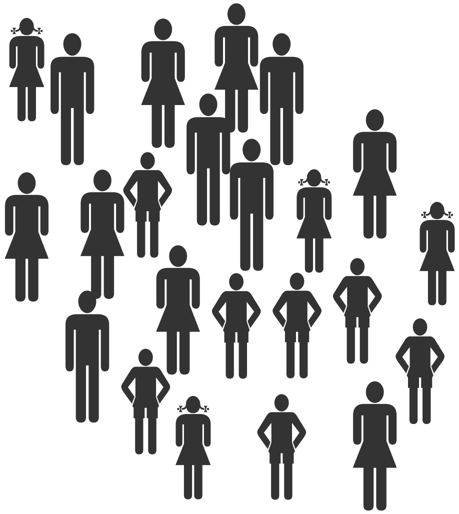


```
In [1]: age = 53

In [2]: age
Out[2]: 53
```





```
In [3]: age_0=53; age_1=44; age_2=18; age_3=16

In [4]: print(age_0,age_1,age_2,age_3)
53 44 18 16
```



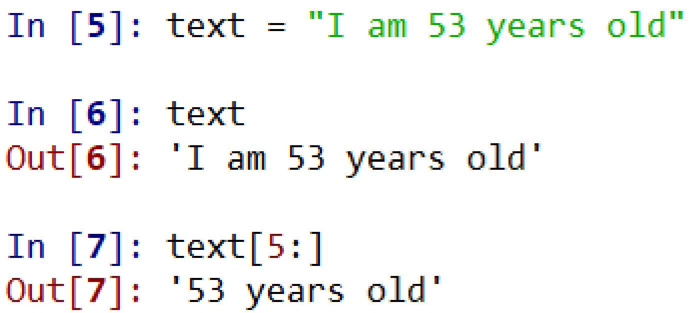
?

We need a data structure for representing a set

Introduction to Data Structures

- We have used the string object: `str`

- `str` is a type of data structure. It’s a set of individual characters.
- We have operators that allow us to access (index) to extract individual characters from a string or cut them to extract substrings.



```
In [5]: text = "I am 53 years old"

In [6]: text
Out[6]: 'I am 53 years old'

In [7]: text[5:]
Out[7]: '53 years old'
```

- We will introduce three data structures:
  - `tuple`, a simple generalization of a string.
  - `sets`, a non-sorted collection of unique objects
  - `lists` and `dictionaries`, very interesting because they are mutable.

Mutable vs Immutable Objects in Python

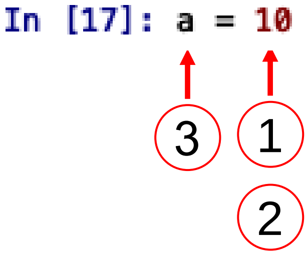
- Most Python objects (`bool, int, float, str` and `tuple`) are **immutable**. This means that, once the object is created and one value has been assigned, this **value cannot be modified**.

- Definition:

  An immutable object is the one whose value cannot be modified.

What does it mean in the computer memory?

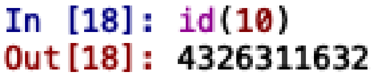
- An object is created, initizalized and stored in the computer memory



```
In [17]: a = 10
```

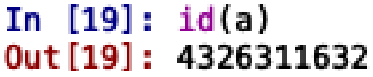
1 An object of type `int` is created with initial value =10

2 This object has an identifier (memory address)



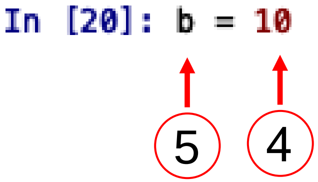
```
In [18]: id(10)
Out[18]: 4326311632
```

3 The name of the variable that refers to this object is a pointer to this memory address



```
In [19]: id(a)
Out[19]: 4326311632
```

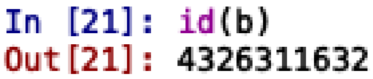
- When we create a new variable to the same object (`int` with initial value of 10)



```
In [20]: b = 10
```

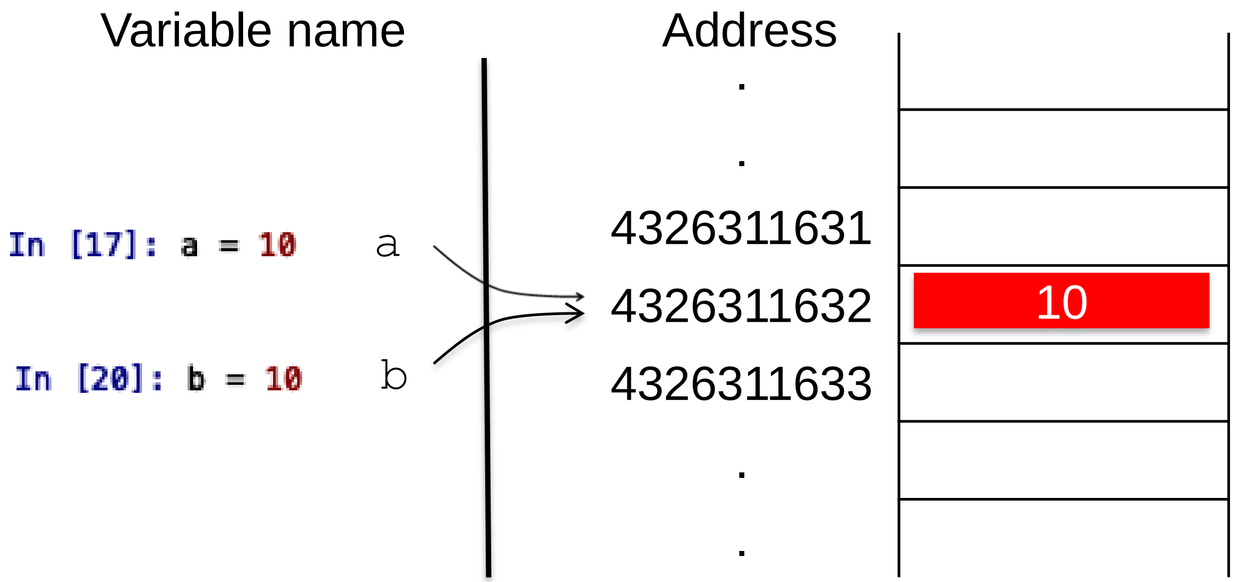
4 A new object of type `int` with value 10 is **not** created, because this object already exists

5 The same memory address is assigned to b



```
In [21]: id(b)
Out[21]: 4326311632
```

What does it mean in the computer memory?



```
In [17]: a = 10
```

```
In [20]: b = 10
```

Variable name

a

b

Address

.

.

4326311631

4326311632

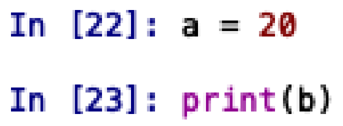
4326311633

.

.

10

- In the hypothetical case in which the object was **mutable**, what would happen if we do:

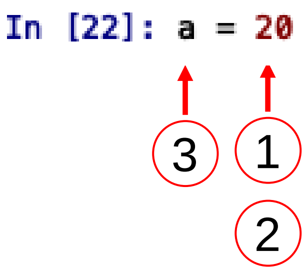


```
In [22]: a = 20

In [23]: print(b)
```

print(b) → 20 !!

Immutables

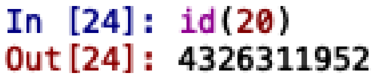
- An object is created, initizalized and stored in the computer memory



```
In [22]: a = 20
```

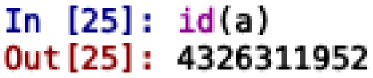
1 An object of type `int` is created with initial value =20

2 This object has an identifier (memory address)

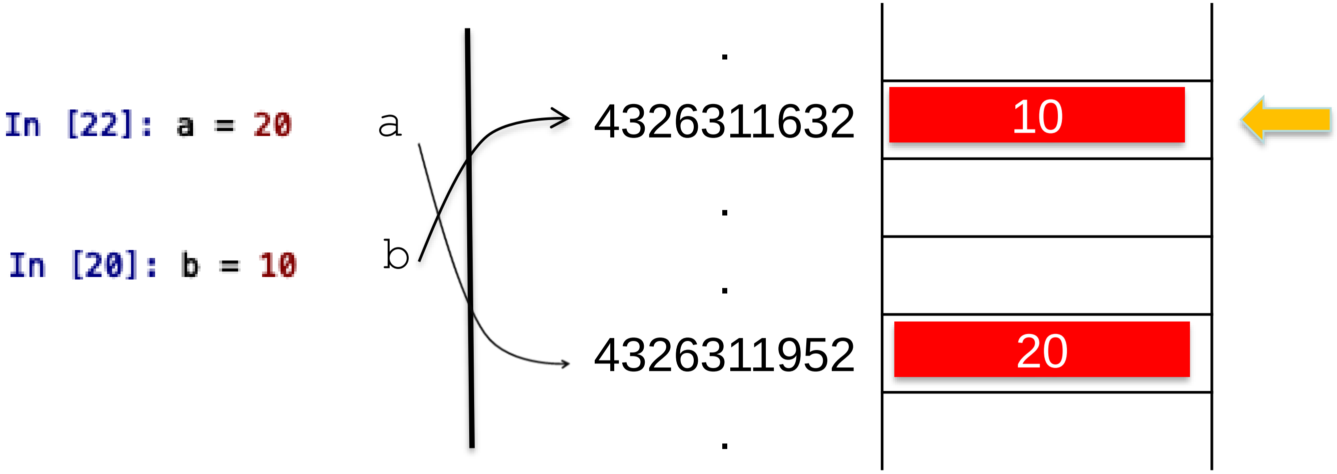


```
In [24]: id(20)
Out[24]: 4326311952
```

3 The name of the variable that refers to this object is a pointer to this memory address



```
In [25]: id(a)
Out[25]: 4326311952
```



```
In [22]: a = 20
```

```
In [20]: b = 10
```

The value in **b** does **NOT** change!!

Need of mutability

- Once the link of the variable to an object is lost, there is no way to return to that object.
- We should create temporal variables to remember their value.
- This implies **wasting memory**, and also, having **cluttered** and **not easy-to-understand** code.

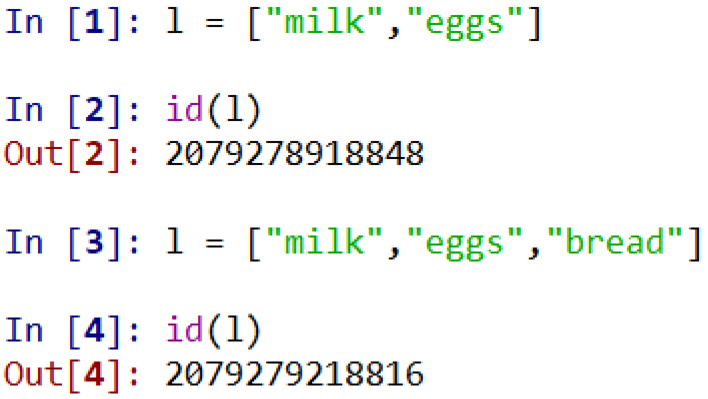
- <u>Definition</u>: A mutable object is an object in which its value **can** change.

Mutability

- Mutable objects are usually objects that store a collection of data, for example, a list.

- Mutable objects behave the same way as immutable ones.

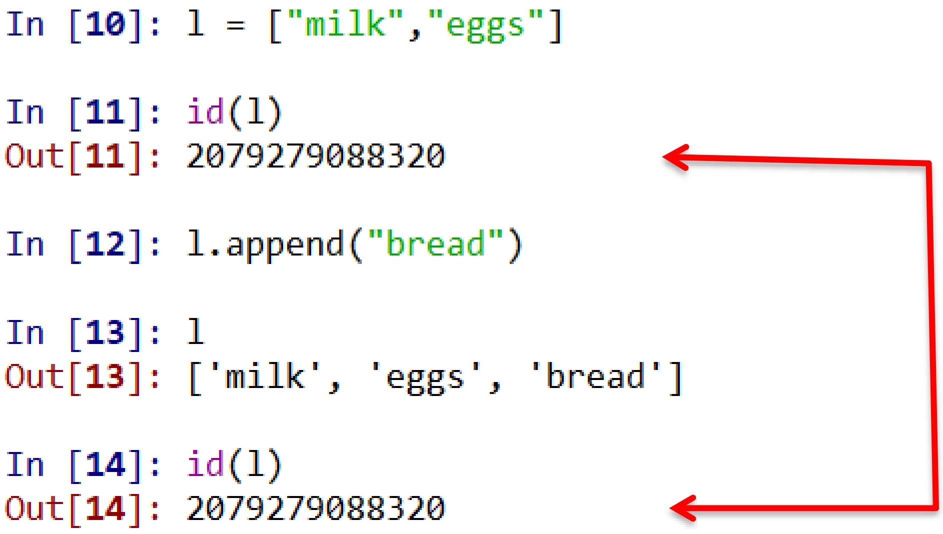
- If a variable (`l`) is a shopping list, when we modify this list to add a new element, the memory address changes, and the pointer to the initial memory address is lost.



```
In [1]: l = ["milk","eggs"]

In [2]: id(l)
Out[2]: 2079278918848

In [3]: l = ["milk","eggs","bread"]

In [4]: id(l)
Out[4]: 2079279218816
```

Mutability

- Instead, we can directly modify the original object without losing the link to the memory address, by using operations that only work on mutable objects.



```
In [10]: l = ["milk","eggs"]

In [11]: id(l)
Out[11]: 2079279088320

In [12]: l.append("bread")

In [13]: l
Out[13]: ['milk', 'eggs', 'bread']

In [14]: id(l)
Out[14]: 2079279088320
```

Mutable
(same id)