LEARN CODING

ale66

github.com/ale66/learn-coding

DATA STRUCTURES

ABSTRACTION

In CS all efforts are made to create -abstractions- of computers/memory that let us program them easily and in a -portable- way

Data structures are presentations of our data that support our coding

SO FAR

- variables as name/type/value triples
- a version of Spreadsheet cells
- *scalar* or *atomic* datum

TODAY

- variables as name/type/structure/values
- a version of spreadsheet columns (or rows)
- multi-dimensional data
- an indexing systems allows to reach each single value

- The way in which the data is organized guides coding and affects computation
- data structures are a *design choice* based on:
    - the nature of the data
    - the processes that need to be performed
- Python offers simple data structures, their adoption affects code readability as well as scalability.

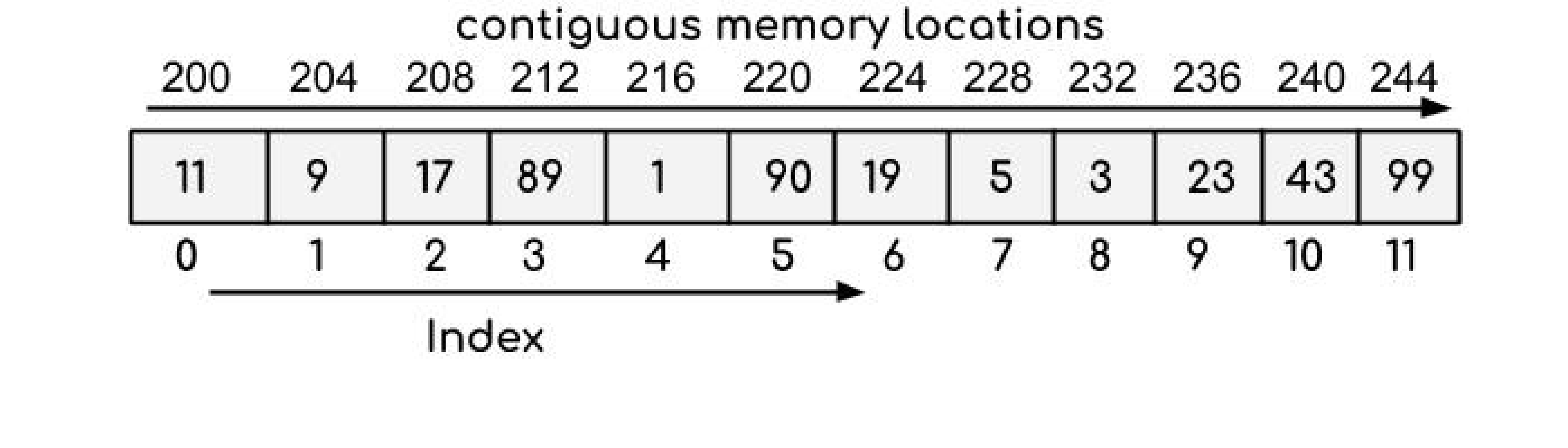
LISTS

THE BASIC STRUCTURE FOR NON-ATOMIC DATA

List: a sequence (order matters) of zero, one or more data items

lists are visualised as

- used to store multiple, oft. homogeneous data in a single variable
- any data type and combinations of data types
- the elements of a list are indexed
- such indexing starts from 0



contiguous memory locations

200	204	208	212	216	220	224	228	232	236	240	244
11	9	17	89	1	90	19	5	3	23	43	99
0	1	2	3	4	5	6	7	8	9	10	11

Index

PROPERTIES OF LISTS

- can be ordered
- values are changeable
- duplicated data items are allowed

QUIZ 1! TRUE OR FALSE?

- A Python list is an ordered and hangeable collection of data where we can only store data of the same data type.
  - **False, we can store data of different types**
- We use the index of a list to access elements, and this index number is the actual address of a memory block.
  - **False, indexes represent the memory addresses, but always start from 0**
- Since lists are indexed, lists can have items with the same value.
  - **True!**

WORK WITH LISTS

A LIST OF STRINGS

A simple list of London football clubs

```python
teams = ['Chelsea FC', 'Arsenal FC', 'Crystal Palace FC', 'West Ham FC']
```

Its lenght is the number of items it contains:

```python
len(teams)
4
```

ACCESS BY INDEX

```
print(teams[0])

Chelsea FC
```

```
print(teams[3])

West Ham FC
```

```python
if (teams[0] == 'Chelsea FC'):

      print('Come on Chelsea!')
```

MUTABLE VALUES

```python
fruitlist = ["apple", "banana", "cherry"]

print(fruitlist)
```

```python
# Update a value
fruitlist[1] = 'blackcurrant'

print(fruitlist)
```

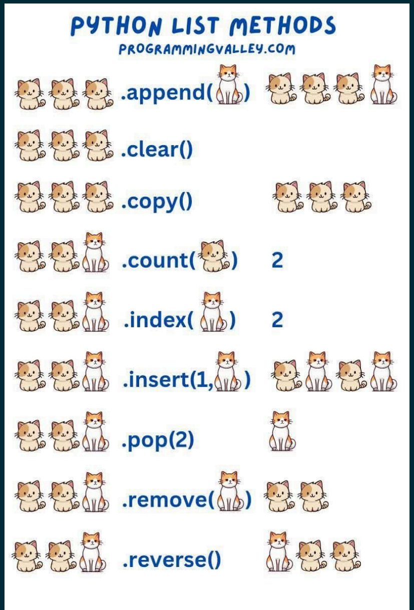
A BRIEF ON *METHODS*

They are automatically attached to a var.

```
my_var.built_in_method()
```

which methods are attached depends on the var. type

For list vars. much is available:



STRINGS

PYTHON STRINGS ARE LISTS

```python
mystring = 'python'
```

p	y	t	h	o	n
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

```python
print(mystring[0])
```

```python
print(mystring[-1])
```

SLICING

We can access arbitrary segments of the list/string:

```
alist = [1, 3, 5, 7, 9, 11]

print(alist[0:2])
```

N.B.: intervals are closed on the left and open on the right!

elements in position 0 & 1 are printed, position 2 ain’t

QUIZ 2! TRUE OR FALSE?

```
mydata = [12, 32, 1, 43, 65]
```

Does

- a. `print(mydata[3])` print 1?
- b. `print(mydata[1]+18` print 50?
- c. `print(mydata[-1])` print 65?
- d. `print(mydata[0:3])` print [12, 32, 1]?

SOLUTIONS

- a: False
- b: True
- c: True
- d: True

ITERATIONS

ITERABLES

Python lists are called *iterables* because they are likely subjects or the repetition (iteration) of a fixed sequence of instructions

```python
# print the content of the string var. vertically:
# each letter on a separate line.
```

```python
for letter in mystring:
    print(letter)
    # a 'newline' is automatically emitted at the end of each print()
```

as with `if`, the 4-spaces indentation defines blocks of code

blocks may be exectued zero, one or many times.

UNHELPFUL...

```
print(mystring[0])
print(mystring[1])
print(mystring[2])
...
```

Inflation! Raise all prices in the menu by 10pc

```python
alist = [2, 3, 5, 7]

# fix me!
for price in alist:
    price = price*1.1
    print(f'The new price is {price}')
```

We need to amend each element of the existing list, as in

```python
  alist[0] = alist[0]*1.1
```

but when to stop?

RANGE

A function that generates the indices needed to amend the list element by element

```
range(5) = 0, 1, 2, 3, 4
```

implictly starts from 0 to enumerate up to

```
range(3, 6) = 3, 4, 5
```

an interval

```
range(3, 8, 2) = 3, 5, 7 # +2 at each step
```

EXAMPLE

```python
alist = [2, 3, 5, 7]

howmany = len(alist)

for i in range(howmany):
    # prices are changed forever
    alist[i] = alist[i]*1.1
```

EXERCISE:

```python
teams = ['Chelsea FC', 'Arsenal FC', 'Crystal Palace FC', 'West Ham FC']
```

print out the club names without the FC suffix

COLLABORATIVE CODING (PAIR CODING)

Get the VS Code extension:

Live Share

(experimental) Sign in with your uni account as if it were Microsoft

create and share tokens to work on the same file (via third-party hosting)

EXERCISE, A

Print available fruits but not bananas

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

hint: use `continue`

EXERCISE, B

Print available fruits but stop as soon as you find bananas

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

hint: use`break