

# Scientific Programming

## Practical 3

---

### Introduction

Luca Bianco - Academic Year 2020-21  
luca.bianco@fmach.it

# Lists

**Ordered** collections of  
(homogeneous) objects

**Mutable** objects

**Defined** using the `[]` and  
items are separated  
by commas

```
my_first_list = [1,2,3]
print("first:" , my_first_list)

my_second_list = [1,2,3,1,3] #elements can appear several times
print("second: ", my_second_list)

fruits = ["apple", "pear", "peach", "strawberry", "cherry"] #elements can be strings
print("fruits:", fruits)

an_empty_list = []
print("empty:" , an_empty_list)

another_empty_list = list()
print("another empty:", another_empty_list)

a_list_containing_other_lists = [[1,2], [3,4,5,6]] #elements can be other lists
print("list of lists:", a_list_containing_other_lists)

my_final_example = [my_first_list, a_list_containing_other_lists]
print("a list of lists of lists:", my_final_example)
```

```
first: [1, 2, 3]
second: [1, 2, 3, 1, 3]
fruits: ['apple', 'pear', 'peach', 'strawberry', 'cherry']
empty: []
another empty: []
list of lists: [[1, 2], [3, 4, 5, 6]]
a list of lists of lists: [[1, 2, 3], [[1, 2], [3, 4, 5, 6]]]
```

# Lists

## Operators and functions

NOTE: as in strings,  
list indexing starts from 0!

Result	Operator	Meaning
bool	=, !=	Check if two lists are equal or different
int	len(list)	Return the length of the list
list	list + list	Concatenate two lists (returns a new list)
list	list * int	Replicate the list (returns a new list)
list	list[int:int]	Extract a sub-list

The whole object  
must be there!

Result	Operator	Meaning
bool	obj in list	Check if an element is present in a list

Lists are **mutable**  
so **now we can**  
**change values!**

Result	Operator	Meaning
obj	list[int]	Read/write an element at a specified index

# Lists

## Operators and functions

NOTE: as in strings,

list indexing starts from 0!

in slicing list[S:E]

S is included

E is excluded

```
A = [1, 2, 3 ]
B = [1, 2, 3, 1, 2]

print("A is a ", type(A))

print(A, " has length: ", len(A))
print("A[0]: ", A[0], " A[1]:", A[1], " A[-1]:", A[-1])

print(B, " has length: ", len(B))
print("Is A equal to B?", A == B)

C = A + [1, 2]
print(C)
print("Is C equal to B?", B == C)
D = [1, 2, 3]*8
print(D)

E = D[12:18] #slicing
print(E)
print("Is A*2 equal to E?", A*2 == E)
```

```
A is a <class 'list'>
[1, 2, 3] has length: 3
A[0]: 1 A[1]: 2 A[-1]: 3
[1, 2, 3, 1, 2] has length: 5
Is A equal to B? False
[1, 2, 3, 1, 2]
Is C equal to B? True
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
[1, 2, 3, 1, 2, 3]
Is A*2 equal to E? True
```

# Lists

## Operators and functions

NOTE: as in strings,  
list indexing starts from 0!

**IN operator:** the whole element must be there!

Lists are **mutable objects** so now we can **change values!**

```
A = [1, 2, 3, 4, 5, 6]
B = [1, 3, 5]
print("A:", A)
print("B:", B)

print("Is B in A?", B in A)
print("A's ID:", id(A))
A[5] = [1,3,5] #we can add elements
print(A)
print("A's ID:", id(A))
print("A has length:", len(A))
print("Is now B in A?", B in A)
```

```
A: [1, 2, 3, 4, 5, 6]
B: [1, 3, 5]
Is B in A? False
A's ID: 140419415368200
[1, 2, 3, 4, 5, [1, 3, 5]]
A's ID: 140419415368200
A has length: 6
Is now B in A? True
```

# Lists

**ERROR: do not exceed boundaries with indexing!**

```
A = [1, 2, 3, 4, 5, 6]
print("A has length:", len(A))

print("First element:", A[0])
print("7th-element: ", A[6])
```

```
A has length: 6
First element: 1
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-5-699e5f04cae0> in <module>()
      3
      4 print("First element:", A[0])
----> 5 print("7th-element: ", A[6])

IndexError: list index out of range
```

# Lists

Slicing can cope with indexes:

```
A = [1, 2, 3, 4, 5, 6]
print("A has length:", len(A))

print("First element:", A[0])
print("last element: ", A[-1])

print("3rd to 10th: ", A[2:10])

print("8th to 11th:", A[7:11])
```


```
A has length: 6
First element: 1
last element: 6
3rd to 10th: [3, 4, 5, 6]
8th to 11th: []
```

# Lists

## Methods

Return	Method	Meaning
None	<code>list.append(obj)</code>	Add a new element at the end of the list
None	<code>list.extend(list)</code>	Add several new elements at the end of the list
None	<code>list.insert(int,obj)</code>	Add a new element at some given position
None	<code>list.remove(obj)</code>	Remove the first occurrence of an element
None	<code>list.reverse()</code>	Invert the order of the elements
None	<code>list.sort()</code>	Sort the elements
int	<code>list.count(obj)</code>	Count the occurrences of an element

can specify  
reverse = True



Note that lists are **mutable objects** and therefore virtually all the previous methods (except *count*) do not have an output value, but **they modify the list**



# Lists

## Methods

```
[1, 2, 3]
[1, 2, 3, 72]
[1, 2, 3, 72, 1, 5, 124, 99]
[99, 124, 5, 1, 72, 3, 2, 1]
[1, 1, 2, 3, 5, 72, 99, 124]
Min value: 1
Max value: 124
Number 1 appears: 2 times
While number 837: 0
```

Done with numbers, let's go strings...

```
['apple', 'banana', 'pineapple', 'cherry', 'pear', 'almond', 'orange']
['pineapple', 'pear', 'orange', 'cherry', 'banana', 'apple', 'almond']
['pineapple', 'pear', 'orange', 'cherry', 'apple', 'almond']
['pineapple', 'pear', 'orange', 'cherry', 'apple', 'wild apple', 'almond']
```

```
#A numeric list
A = [1, 2, 3]
print(A)
print("A has id:", id(A))
A.append(72) #appends one and only one object
print(A)
print("A has id:", id(A))
A.extend([1, 5, 124, 99]) #adds all these objects, one after the other.
print(A)
A.reverse() #NOTE: NO RETURN VALUE!!!
print(A)
A.sort()
print(A)
print("Min value: ", A[0]) # In this simple case, could have used min(A)
print("Max value: ", A[-1]) #In this simple case, could have used max(A)
print("Number 1 appears:", A.count(1), " times")
print("While number 837: ", A.count(837))

print("\nDone with numbers, let's go strings...\n")
#A string list
fruits = ["apple", "banana", "pineapple", "cherry", "pear", "almond", "orange"]
#Let's get a reverse lexicographic order:
print(fruits)
fruits.sort()
fruits.reverse() # equivalent to: fruits.sort(reverse=True)
print(fruits)
fruits.remove("banana")
print(fruits)
fruits.insert(5, "wild apple") #put wild apple after apple.
print(fruits)
print("\nSorted fruits:")
fruits.sort() # does not return anything. Modifies list!
print(fruits)
```

# Lists

## Methods

lists **are mutable objects**  
and therefore **virtually all**  
**the previous methods**  
**(except count) do not have**  
**an output value:**

```
A = ["A", "B", "C"]  
  
print("A:", A)  
  
A_new = A.append("D")  
  
print("A:", A)  
  
print("A_new:", A_new)  
  
A: ['A', 'B', 'C']  
A: ['A', 'B', 'C', 'D']  
A_new: None
```



Return	Method	Meaning
None	<code>list.append(obj)</code>	Add a new element at the end of the list
None	<code>list.extend(list)</code>	Add several new elements at the end of the list
None	<code>list.insert(int,obj)</code>	Add a new element at some given position
None	<code>list.remove(obj)</code>	Remove the first occurrence of an element
None	<code>list.reverse()</code>	Invert the order of the elements
None	<code>list.sort()</code>	Sort the elements
int	<code>list.count(obj)</code>	Count the occurrences of an element

# Lists

## Some important things on lists

### 1. append is different from extend

```
A = [1, 2, 3]

A.extend([4, 5])
print(A)
B = [1, 2, 3]
B.append([4, 5])
print(B)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, [4, 5]]
```

### 2. to remove an object it must exist

```
A = [1, 2, 3]
A.remove(2)
print(A)
A.remove(7)
```

```
[1, 3]
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-9-bdf156ee14f6> in <module>()
      2 A.remove(2)
      3 print(A)
----> 4 A.remove(7)

ValueError: list.remove(x): x not in list
```

The correct way would be to test if 7 is present in A (we will do that when we know the syntax of if statements)

# Lists

## Some important things on lists

### 3. a list is sortable if all its elements are (i.e. it is homogeneous)

```
A = [4,3, 1,7, 2]
print(A)
A.sort()
print(A)
A.append("banana")
print(A)
A.sort()
print(A)
```

```
[4, 3, 1, 7, 2]
[1, 2, 3, 4, 7]
[1, 2, 3, 4, 7, 'banana']
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-1-91b77adb823f> in <module>
      5 A.append("banana")
      6 print(A)
----> 7 A.sort()
      8 print(A)
```

```
TypeError: '<' not supported between instances of 'str' and 'int'
```

# Lists

## REMEMBER:

Lists are **MUTABLE** objects...  
... hence they hold references  
to objects rather than objects.

```
A = ["hi", "there"]
B = A
print("A:", A)
print("B:", B)
A.extend(["from", "python"])
print("A now: ", A)
print("B now: ", B)

print("\n---- copy example -----")
#Let's make a distinct copy of A.
C = A[:] #all the elements of A have been copied in C
print("C:", C)
A[3] = "java"
print("A now:", A)
print("C now:", C)

print("\n---- be careful though -----")
#Watch out though that...
D = [A, A]
E = D[:]
print("D:", D)
print("E:", E)

D[0][0] = "hello"
print("D now:", D)
print("E now", E)
```

```
A: ['hi', 'there']
B: ['hi', 'there']
A now: ['hi', 'there', 'from', 'python']
B now: ['hi', 'there', 'from', 'python']

---- copy example -----
C: ['hi', 'there', 'from', 'python']
A now: ['hi', 'there', 'from', 'java']
C now: ['hi', 'there', 'from', 'python']

---- be careful though -----
D: [['hi', 'there', 'from', 'java'], ['hi', 'there', 'from', 'java']]
E: [['hi', 'there', 'from', 'java'], ['hi', 'there', 'from', 'java']]
D now: [['hello', 'there', 'from', 'java'], ['hello', 'there', 'from', 'java']]
E now [['hello', 'there', 'from', 'java'], ['hello', 'there', 'from', 'java']]
```

# The split method. String → List

## Syntax:

**LIST = str.split(str)**



string to be split



split at characters

```
text = "This is my sentence. How many words have I written?"  
words = text.split(' ')  
  
print(text)  
print(words)  
print("\nThe sentence contains ", len(words), "words")
```

This is my sentence. How many words have I written?

['This', 'is', 'my', 'sentence.', 'How', 'many', 'words', 'have', 'I', 'written?']

The sentence contains 10 words



# The split method

**Example** Recall the protein seen in the previous practical:

```
chain_a = ""SSSVPSQKTYQGSYGFR LGFLHSGTAKSVTCTYSPALNKM  
FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMTEVV  
RRCPPHHERCSDSDGLAPPQHLIRVEGNLRVEYLDDRNTFR  
HSVVVPYEPPEVGSDCTTIHYNMCMSSCMGGMNRRPILT  
IITLEDSSGNLLGRNSFEVRVCACPGDRRTEENLRKKG EPHHELPPGSTKRALPNNT""
```

how can we split it into several lines?

```
chain_a = ""SSSVPSQKTYQGSYGFR LGFLHSGTAKSVTCTYSPALNKM  
FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMTEVV  
RRCPPHHERCSDSDGLAPPQHLIRVEGNLRVEYLDDRNTFR  
HSVVVPYEPPEVGSDCTTIHYNMCMSSCMGGMNRRPILT  
IITLEDSSGNLLGRNSFEVRVCACPGDRRTEENLRKKG  
EPHHELPPGSTKRALPNNT""
```

```
lines = chain_a.split('\n')  
print("Original sequence:")  
print(chain_a, "\n") #some spacing to keep things clear  
print("line by line:")  
# write the following and you will appreciate loops! :-)  
print("1st line:", lines[0])  
print("2nd line:", lines[1])  
print("3rd line:", lines[2])  
print("4th line:", lines[3])  
print("5th line:", lines[4])  
print("6th line:", lines[5])
```

```
print("\nSplit the 1st line in correspondence of FRL:\n", lines[0].split("FRL"))
```

Original sequence:

```
SSSVPSQKTYQGSYGFR LGFLHSGTAKSVTCTYSPALNKM  
FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMTEVV  
RRCPPHHERCSDSDGLAPPQHLIRVEGNLRVEYLDDRNTFR  
HSVVVPYEPPEVGSDCTTIHYNMCMSSCMGGMNRRPILT  
IITLEDSSGNLLGRNSFEVRVCACPGDRRTEENLRKKG  
EPHHELPPGSTKRALPNNT
```

line by line:

```
1st line: SSSVPSQKTYQGSYGFR LGFLHSGTAKSVTCTYSPALNKM  
2nd line: FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMTEVV  
3rd line: RRCPPHHERCSDSDGLAPPQHLIRVEGNLRVEYLDDRNTFR  
4th line: HSVVVPYEPPEVGSDCTTIHYNMCMSSCMGGMNRRPILT  
5th line: IITLEDSSGNLLGRNSFEVRVCACPGDRRTEENLRKKG  
6th line: EPHHELPPGSTKRALPNNT
```

```
Split the 1st line in correspondence of FRL:  
['SSSVPSQKTYQGSYG', 'GFLHSGTAKSVTCTYSPALNKM']
```

# The split method

**Example** Recall the protein seen in the previous practical:

```
chain_a = ""SSSVPSQKTYQGSYGFR LGFLHSGTAKSVTCTYSPALNKM  
FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMT EVV  
RRCPPHHERCSDSDGLAPPQHLIRVEGNLRVEYLD DRNTFR  
HSVVVPYEPPEVGSDCTTIHYNM CNSSCMGGMNRRPILT  
IITLEDSSGNLLGRNSFEVRVCACPGDRDRTEENLRKKG EPHHELPPGSTKRALPNNT""
```

how can we split it into several lines?

where is FRL gone?



```
chain_a = ""SSSVPSQKTYQGSYGFR LGFLHSGTAKSVTCTYSPALNKM  
FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMT EVV  
RRCPPHHERCSDSDGLAPPQHLIRVEGNLRVEYLD DRNTFR  
HSVVVPYEPPEVGSDCTTIHYNM CNSSCMGGMNRRPILT  
IITLEDSSGNLLGRNSFEVRVCACPGDRDRTEENLRKKG  
EPHHELPPGSTKRALPNNT""
```

```
lines = chain_a.split('\n')  
print("Original sequence:")  
print(chain_a, "\n") #some spacing to keep things clear  
print("line by line:")  
# write the following and you will appreciate loops! :-)  
print("1st line:", lines[0])  
print("2nd line:", lines[1])  
print("3rd line:", lines[2])  
print("4th line:", lines[3])  
print("5th line:", lines[4])  
print("6th line:", lines[5])
```

```
print("\nSplit the 1st line in correspondence of FRL:\n", lines[0].split("FRL"))
```

Original sequence:

```
SSSVPSQKTYQGSYGFR LGFLHSGTAKSVTCTYSPALNKM  
FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMT EVV  
RRCPPHHERCSDSDGLAPPQHLIRVEGNLRVEYLD DRNTFR  
HSVVVPYEPPEVGSDCTTIHYNM CNSSCMGGMNRRPILT  
IITLEDSSGNLLGRNSFEVRVCACPGDRDRTEENLRKKG  
EPHHELPPGSTKRALPNNT
```

line by line:

```
1st line: SSSVPSQKTYQGSYGFR LGFLHSGTAKSVTCTYSPALNKM  
2nd line: FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMT EVV  
3rd line: RRCPPHHERCSDSDGLAPPQHLIRVEGNLRVEYLD DRNTFR  
4th line: HSVVVPYEPPEVGSDCTTIHYNM CNSSCMGGMNRRPILT  
5th line: IITLEDSSGNLLGRNSFEVRVCACPGDRDRTEENLRKKG  
6th line: EPHHELPPGSTKRALPNNT
```

```
Split the 1st line in correspondence of FRL:  
['SSSVPSQKTYQGSYG', 'GFLHSGTAKSVTCTYSPALNKM']
```



# The join method. List → String

**Example** Given the list ['Oct', '5', '2018', '15:30'], let's combine all its elements in a string joining the elements with a dash ("-") and print them. Let's finally join them with a tab ("\t") and print them.

Syntax:

str.join(list)



elements to join

string used to join  
them

```
vals = ['Oct', '5th', '2018', '15:30']  
print(vals)  
myStr = "-".join(vals)  
print("\n" + myStr)  
myStr = "\t".join(vals)  
print("\n" + myStr)
```

```
['Oct', '5th', '2018', '15:30']
```

```
Oct-5th-2018-15:30
```

```
Oct      5th      2018      15:30
```

# Tuples

Tuples are the IMMUTABLE  
version of lists  
(ordered sequence of objects)

```
first_tuple = (1,2,3)
print(first_tuple)

second_tuple = (1,) #this contains one element only, but we need the comma!
var = (1) #This is not a tuple!!!
print(second_tuple, " type:", type(second_tuple))
print(var, " type:", type(var))
empty_tuple = () #fairly useless
print(empty_tuple)
third_tuple = ("January", 1, 2007) #heterogeneous info
print(third_tuple)

days = (third_tuple, ("February", 2, 1998), ("March", 2, 1978), ("June", 12, 1978))
print(days, "\n")

#Remember tuples are immutable objects...
print("Days has id: ", id(days))
days = ("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")
#...hence reassignment creates a new object
print("Days now has id: ", id(days))
```

---

```
(1, 2, 3)
(1,) type: <class 'tuple'>
1 type: <class 'int'>
()
('January', 1, 2007)
(('January', 1, 2007), ('February', 2, 1998), ('March', 2, 1978), ('June', 12, 1978))

Days has id: 140419415813880
Days now has id: 140419416147240
```

# Tuples

Tuples are the IMMUTABLE

version of lists

(ordered sequence of objects)

Why? Tuples can be used as keys of dictionary,  
whereas lists cannot

```
a = [1, 2, [1,2,3]] # a list
b = (1, 2, [1,2,3]) # a tuple
print("a:", a)
print("b:", b)
print("")
print("a[0]:", a[0], "b[0]:", b[0])
print("a[2]:", a[2], "b[2]:", b[2])
```

```
a: [1, 2, [1, 2, 3]]
b: (1, 2, [1, 2, 3])
```

```
a[0]: 1 b[0]: 1
a[2]: [1, 2, 3] b[2]: [1, 2, 3]
```

# Tuples

Tuples are the IMMUTABLE  
version of lists  
(ordered sequence of objects)

... and do not support item  
assignment

```
a = [1, 2, [1,2,3]] # a list
b = (1, 2, [1,2,3]) # a tuple
print("a:", a)
print("b:", b)
print("")
print("a[0]:", a[0], "b[0]:", b[0])
print("a[2]:", a[2], "b[2]:", b[2])
print("")
a[1] = [7,8,9]
print(a)
b[1] = [7,8,9]
print(b)
```

```
a: [1, 2, [1, 2, 3]]
b: (1, 2, [1, 2, 3])

a[0]: 1 b[0]: 1
a[2]: [1, 2, 3] b[2]: [1, 2, 3]

[1, [7, 8, 9], [1, 2, 3]]
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-2-549ff0d2c315> in <module>
      9 a[1] = [7,8,9]
     10 print(a)
--> 11 b[1] = [7,8,9]
     12 print(b)
     13
```

TypeError: 'tuple' object does not support item assignment

# Tuples

Functions

working as in lists...

Result	Operator	Meaning
bool	<code>=, !=</code>	Check if two tuples are equal or different
int	<code>len(tuple)</code>	Return the length of the tuple
tuple	<code>tuple + tuple</code>	Concatenate two tuples (returns a new tuple)
tuple	<code>tuple * int</code>	Replicate the tuple (returns a tuple)
tuple	<code>tuple[int]</code>	Read an element of the tuple
tuple	<code>tuple[int:int]</code>	Extract a sub-tuple

# Tuples

## Functions

```
practical1 = ("Wednesday", "23/09/2020")
practical2 = ("Monday", "28/09/2020")
practical3 = ("Wednesday", "30/09/2020")

#A tuple containing 3 tuples
lectures = (practical1, practical2, practical3)
#One tuple only
mergedLectures = practical1 + practical2 + practical3

print("The first three lectures:\n", lectures, "\n")
print("mergedLectures:\n", mergedLectures)

#This returns the whole tuple
print("1st lecture was on: ", lectures[0], "\n")
#2 elements from the same tuple
print("1st lecture was on ", mergedLectures[0], ", ", mergedLectures[1], "\n")
# Return type is tuple!
print("3rd lecture was on: ", lectures[2])
#2 elements from the same tuple returned in tuple
print("3rd lecture was on ", mergedLectures[4:], "\n")
```

The first three lectures:

```
(( 'Wednesday', '23/09/2020'), ('Monday', '28/09/2020'), ('Wednesday', '30/09/2020'))
```

mergedLectures:

```
('Wednesday', '23/09/2020', 'Monday', '28/09/2020', 'Wednesday', '30/09/2020')
```

1st lecture was on: ('Wednesday', '23/09/2020')

1st lecture was on Wednesday , 23/09/2020

3rd lecture was on: ('Wednesday', '30/09/2020')

3rd lecture was on ('Wednesday', '30/09/2020')

# Tuples

## Methods

working as in lists...

Return	Method	Meaning
int	<code>tuple.count(obj)</code>	Count the occurrences of an element
int	<code>tuple.index(obj)</code>	Return the index of the first occurrence of an object



# Tuples

## Methods

```
practical1 = ("Wednesday", "23/09/2020")
practical2 = ("Monday", "28/09/2020")
practical3 = ("Wednesday", "30/09/2020")
```

```
mergedLectures = practical1 + practical2 + practical3 #One tuple only
print(mergedLectures.count("Wednesday"), " lectures were on Wednesday")
print(mergedLectures.count("Monday"), " lecture was on Monday")
print(mergedLectures.count("Friday"), " lectures was on Friday")

print("Index:", practical2.index("Monday"))
#You cannot look for an element that does not exist
print("Index:", practical2.index("Wednesday"))
```

```
2 lectures were on Wednesday
1 lecture was on Monday
0 lectures was on Friday
Index: 0
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-19-20063b595bbc> in <module>
    10
    11 print("Index:", practical2.index("Monday"))
--> 12 print("Index:", practical2.index("Wednesday"))
    13
```

```
ValueError: tuple.index(x): x not in tuple
```



Questions ?



<https://qcbsciprolab2020.readthedocs.io/en/latest/practical3.html>

Go quickly  
through the  
text and do  
the exercises  
at the end

### Exercises

1. Given the following text string:

```
"""this is a text  
string on  
several lines that does not say anything."""
```

a. print it; b) print how many lines, words and characters it contains. Finally, c) sort the words alphabetically and print the first and the last in lexicographic order.

Show/Hide Solution

2. The variant calling format (**VCF**) is a format to represent structural variants of genomes (i.e. SNPs, insertions, deletions). Each line of this format represents a variant, every piece of information within a line is separated by a tab (\t in python). The first 5 fields of this format report the chromosome (chr), the position (pos), the name of the variant (name), the reference allele (REF) and the alternative allele (ALT). Assuming to have a variable VCF defined containing the following three lines (representing three SNPs):

```
VCF = """MDC000001.124\t7112\tFB_AFFY_0000024\tG\tA  
MDC000002.320\t941\tFB_AFFY_0000144\tC\tT  
MDC000004.272\t2015\tFB_AFFY_0000222\tG\tA"""
```

1. Store these three variants as a list of lists, where each one of the fields is kept separate (e.g. the list should be similar to: `[[chr1, pos1, name1, ref1, alt1], [chr2, pos2, name2, ref2, alt2], ...]` where all the elements are as specified in the string VCF (note that "... " means that the list is not complete).

2. Print each variant changing its format in: `"name|chr|pos|REF|ALT"`.