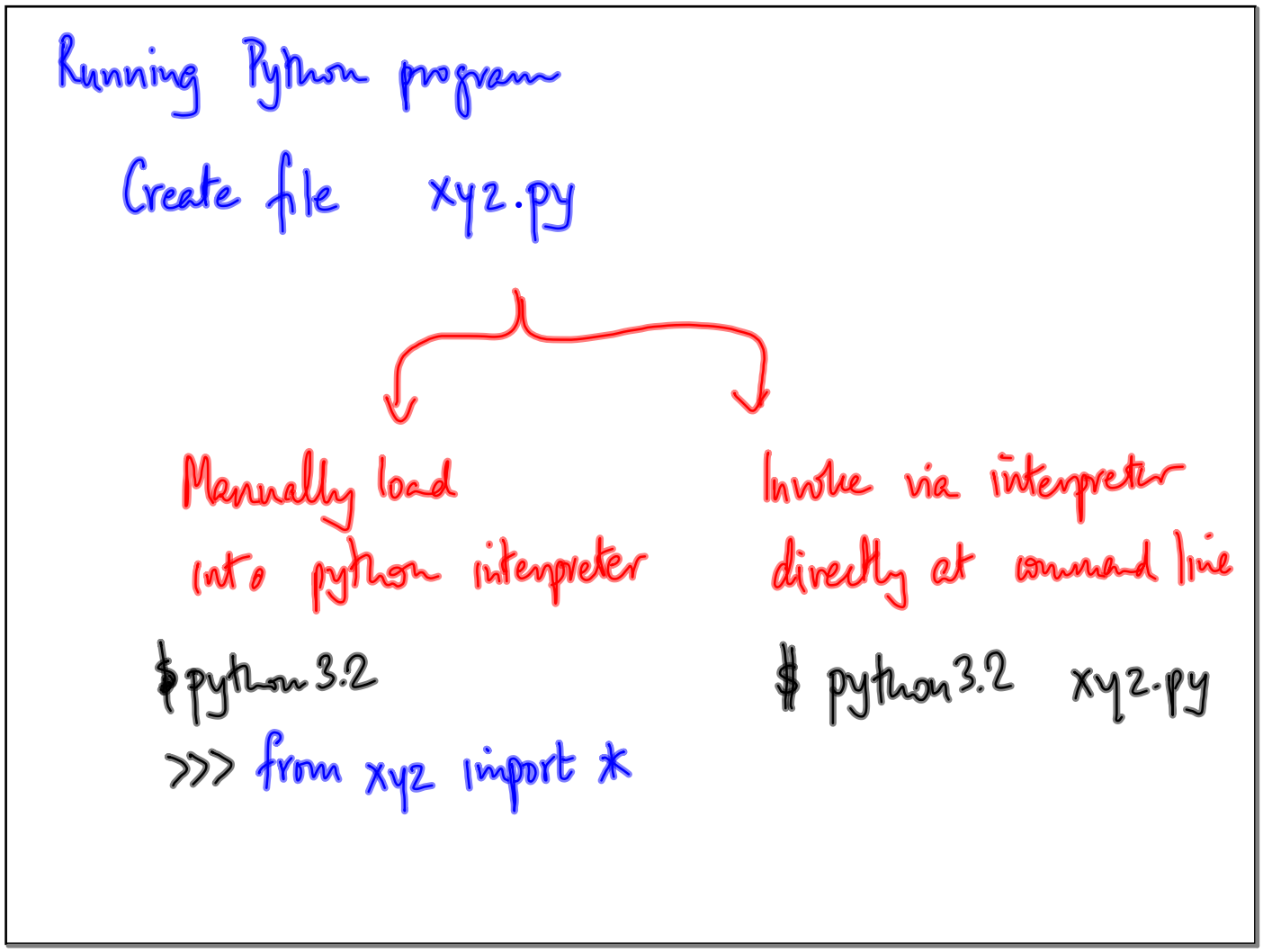
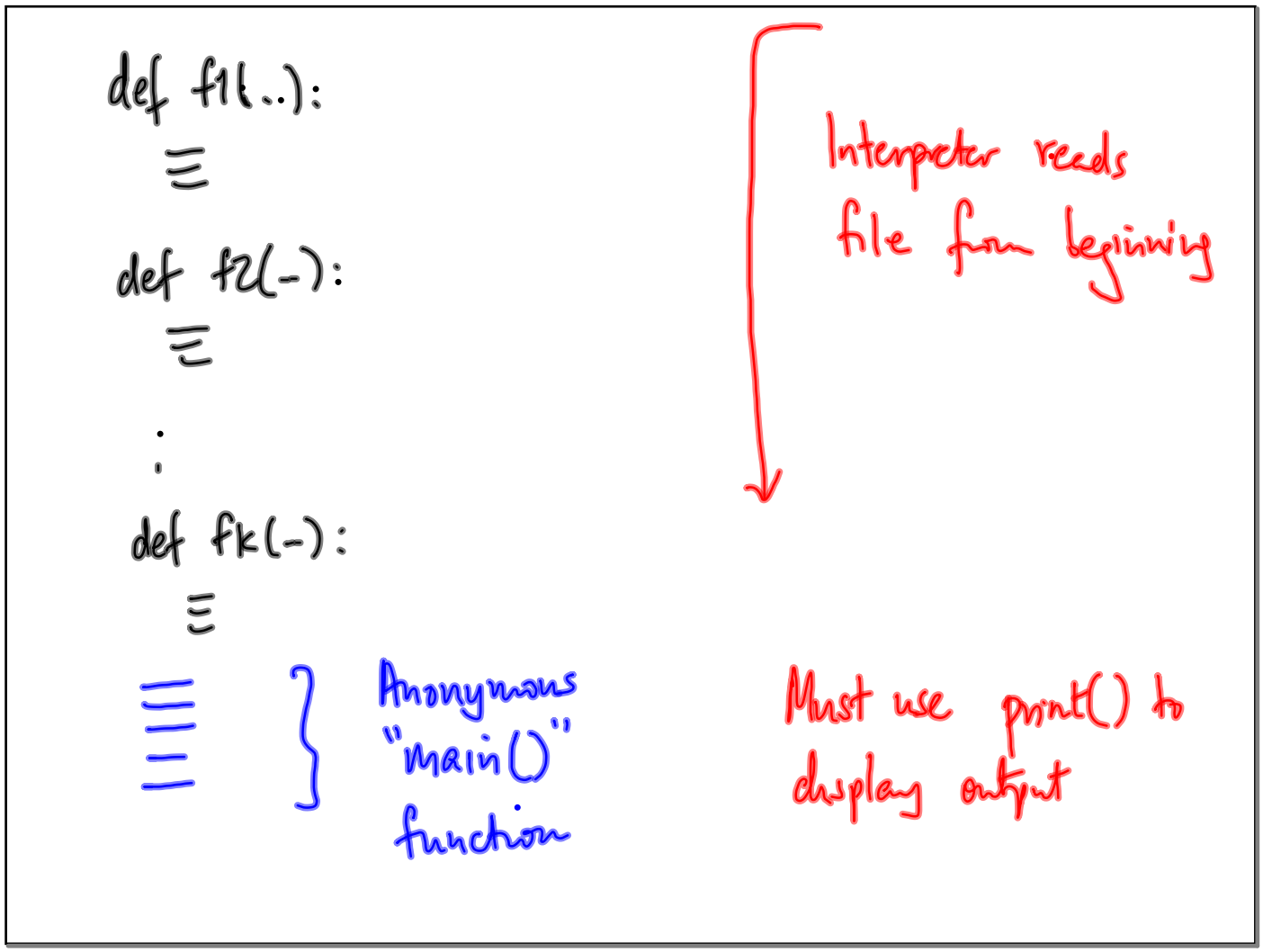

Running Python program
Create file xyz.py
Manually load into python interpreter
$ python3.2
>>> from xyz import *
Invoke via interpreter directly at command line
$ python3.2 xyz.py


def f1(..):
def f2(..):
def fk(..):
Anonymous "main()" function
Interpreter reads file from beginning
Must use print() to display output

Another option

```
def f1()
    :

def fk()
    ≡

def main()
    ≡

main()
```

```
l: for i in ___:
       for j in ___:
           :
           break out of both loops
           break(l)
```

ALAS, not in Python

List manipulation

In place update

`l[i] = v`

`l[i:j] = [..]`

`l.append(v)`

`l.extend(l2)`

`l.insert(p,x)`

Fresh object created

`l = l[0:i]+[v]+..`

`l = l+l2`

inserts x before position p

`l.insert(0,x)` ≡ `x:l`

x is y — True if x & y refer to same object in memory

x is y ⇒ x == y

but not vice versa

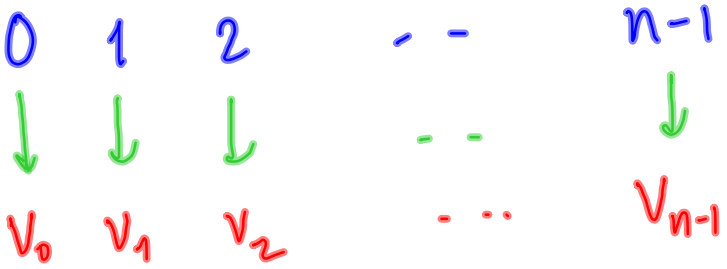
Some more list functions

`x in l`	Is `x` an element of `l`? elem x l
`l.index(x)`	first position where `x` occurs error if `x` not in `l`
`l.remove(x)`	removes all copies of `x` from `l` error if `x` not in `l`
`del(l[i])`	delete position `i`

↳ more generally, `del(x)` unsets `x`

New type of list:

A list is an association from positions to values



More generally, associate values to "keys"

Python calls this a dictionary

Setting up a dictionary

```
name = { key1 : value1 , key2 : value2 -- , keyn : valn }

scorecard = { 'Sehwag' : 0 , 'Gambhir' : 0 , -- }

openerscore = scorecard ['Sehwag'] +
              scorecard ['Gambir']
```

Dictionaries are mutable

Functions can update them in place

Keys, values can be of mixed types

Keys must be immutable

Python optimizes dictionary structure for key based access

Order of items may change

If d is already a dictionary, add an entry

```
d[newkey] = newvalue
```

To start with

```
d = {}
```

sets up an empty dictionary

```
d = {}
for i in [...]:
    d[i] = f(i)
```

Do something for all elements in a dictionary

d.keys() – (not quite a) list of all the keys

d.values() – " " values

```
for i in d.keys():
    ≡
```

Can sort a list in place using l.sort()

ascendig

Suppose we want to process a dictionary in ascending order of keys

d.keys() is not a list

```
l = list(d.keys())
for i in l.sort():
    ≡
```

Removing a value

del (d[k])

Suppose we want to do the following:

Given a key k and dictionary d

if d[k] exists

increment d[k]

else

create d[k] = 0

Solution 1:

```
if k in list(d.keys()):
    d[k] = d[k] + 1
else:
    d[k] = 0
```

Another approach

Try to increment d[k]

If this fails; set d[k] = 0

↓

need to decide & recover from errors within code

Exception handling