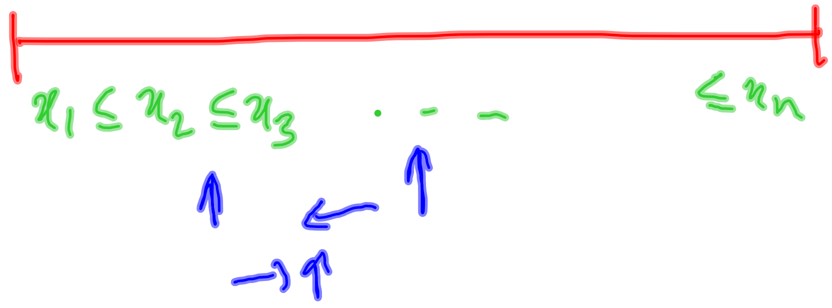
Quiz: Binary search

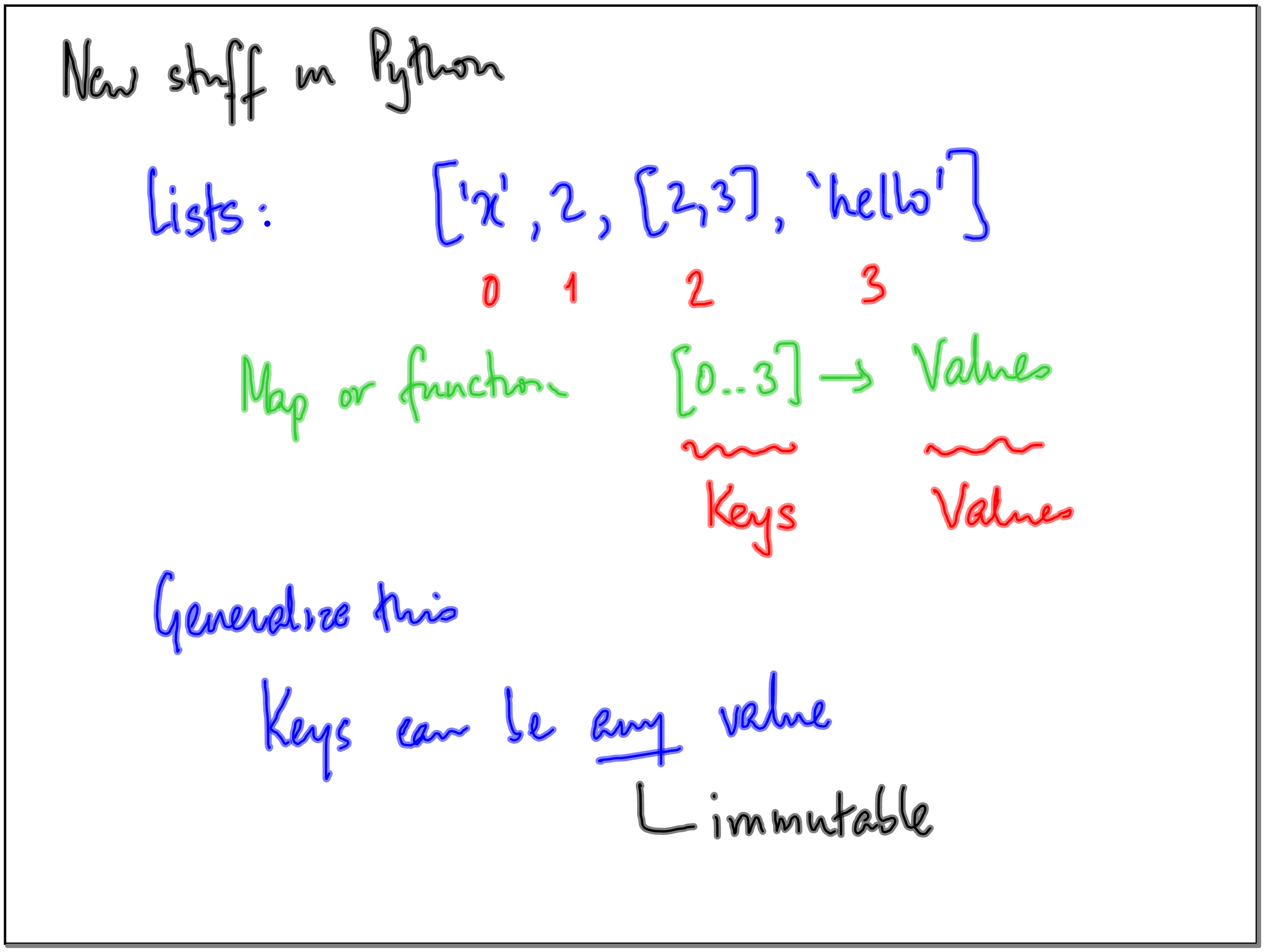


$T(n) = 1 + T(n/2) \rightarrow$ Expand $\log_2 n$ times till we reach $T(1)$

$T(1) = 1$

$$T(n) = \underbrace{1+1+1+\dots+}_{\log n \text{ times}} T(1)$$

$= O(\log_2 n)$



New stuff in Python

Lists: $\underset{0}{\text{'x'}},\ \underset{1}{2},\ \underset{2}{[2,3]},\ \underset{3}{\text{'hello'}}$ ['x', 2, [2,3], 'hello']

Map or function $\underbrace{[0..3]}_{\text{Keys}} \to \underbrace{\text{Values}}_{\text{Values}}$

Generalize this

Keys can be <u>any</u> value

└ immutable

Cricket scorecard

$$\begin{array}{ccccccccccccc} [& 0, & 0, & 0, & 3, & 2, & 22, & 33, & 3, & 4, & 3, & 3 &] \\ & | & | & | & | & | & | & | & | & | & | & | & \\ & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \end{array}$$

Instead:

$$\text{'Sehwag'} \longrightarrow 0$$

$$\vdots$$

Dictionary in Python

```
Scorecard = {'Sehwag':0, 'Gambhir':0, ..  }

scorecard['Raina'] ---> 22
```

↑ key instead of position

Add values to a dictionary

```
scorecard['Trott'] = 1567
```

Update value if key exists, else add new key-value pair

Keys can be any immutable value

```
d = {}    # d is an empty dictionary

d[0] = 0
d[1] = 0
  ⋮
        {0:0, 1:0, ... }
```

Not be confused with

```
d = []
d[0] = 0        # oops, IndexError
```

Only constraint on keys: immutable

```
d[(2,3)] = True
```

CommonDivisors = {}

CommonDivisors[(6,9)] = [1,3]

{(6,9): [1,3]}

CommonDivisors[(6,9)][1] $\rightsquigarrow$ 3

Read a sequence of the form

$name_1$, $score_1$

$name_2$, $score_2$

Some names will repeat

Eventually want a dictionary with cumulative scores for each name

Dictionaries are optimized to look up values by key

Order in which key-value pairs are entered is not guaranteed to be preserved

---

Back to our problem

How to determine if a given key exists?

Examine all keys of a dictionary?

```
if new name is in the list of keys
    add score to current total
else
    create a new entry
```

d is a dictionary

d.keys() "enumerates" the keys

Make it a list by list(d.keys())

Like range(m,n)

```
for k in range(m,n):
    ⋮
```

```
for k in d.keys():
    ⋮
```

Can check for a value in a list

```
x in l
name in list(d.keys())
```

```
score = {'Sehwag':0, 'Dravid':55}

'Sehwag' in list(score.keys()) → True

name = 'Dravid'

name in list(score.keys()) → True

    if name in list(score.keys()):
        score[name] = score[name] + newscore
    else:
        score[name] = newscore
```

Nested Dictionaries

```
score {'innings1': {'Sehwag':0, 'Dravid':123},
       'innings2': {'Sehwag':0, 'Dravid':12}}
```

`score['innings2']['Dravid']` $\dashrightarrow$ 12

If you want to process keys in sorted order, assuming they can be sorted

```
k = list(d.keys())
k.sort()
for n in k:
    ⋮
```

name = 'Zaheer'

x = 2

score [name] = x $\rightsquigarrow$ creates entry 'Zaheer' : 2

Removing an entry from dictionary?

del (score ['Zaheer'])

score.values() – extracts values, like keys

```
Problem: If key exists:
             increment/update
         else:
             create
```

Option 1: Examine d.keys()

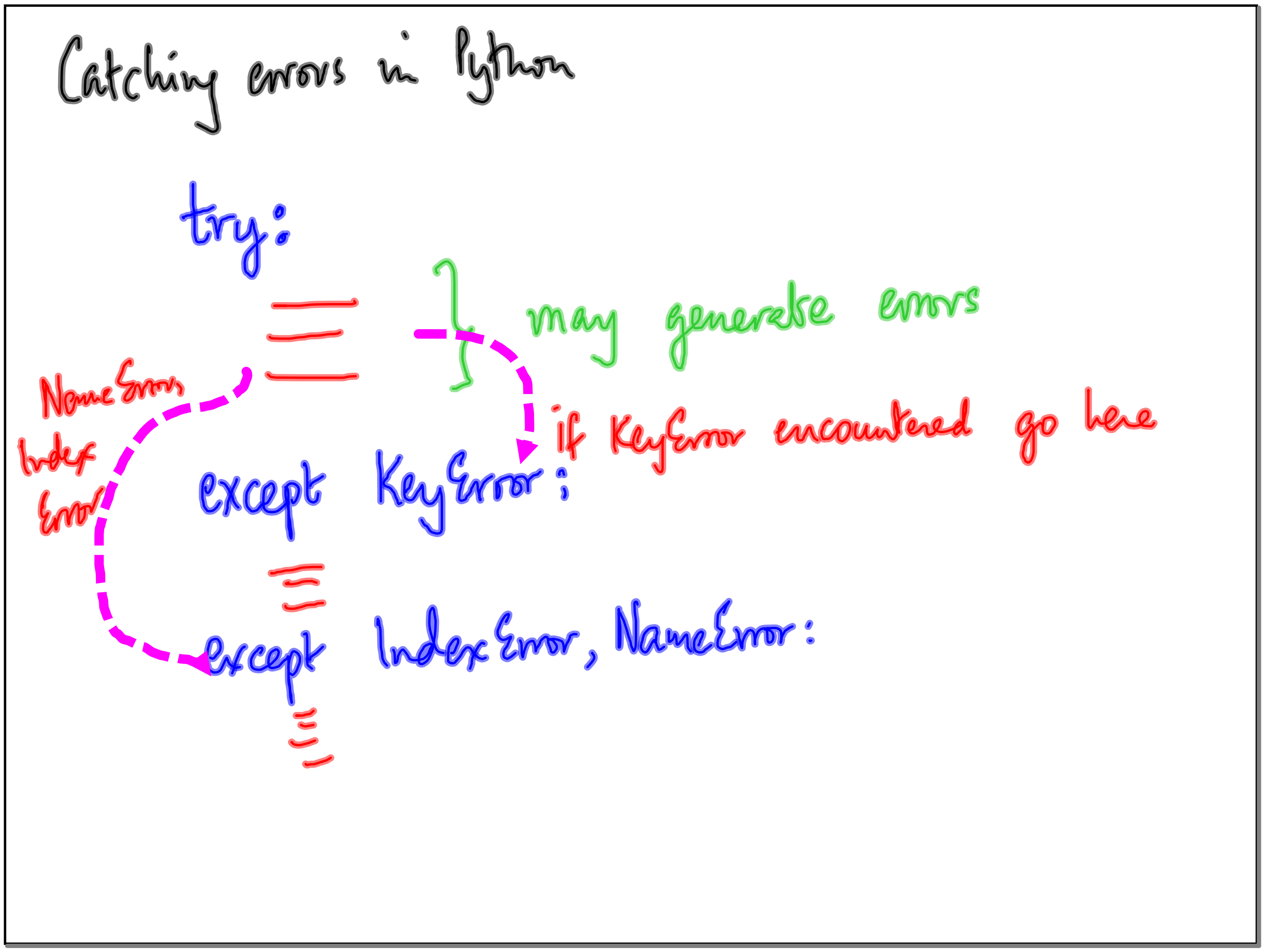
Option 2: Check if accessing the key gives an error

try to update the value for this key
if this fails (error, no key exists), create
new key-value pair

How to process errors within your program?

Observe: Each error has a "name"

IndexError, KeyError, NameError

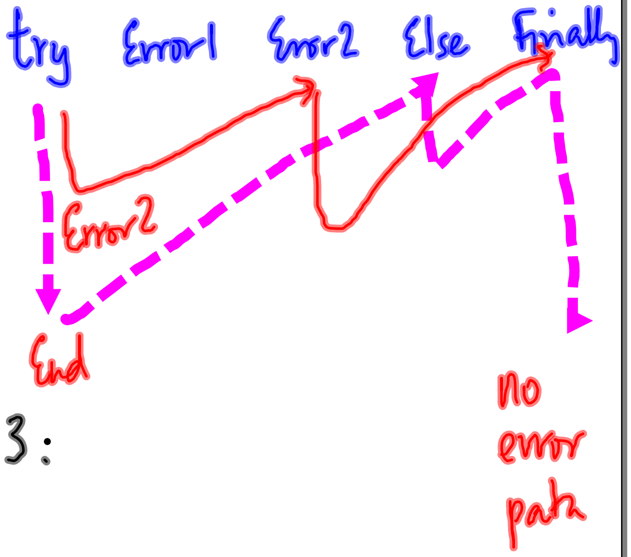


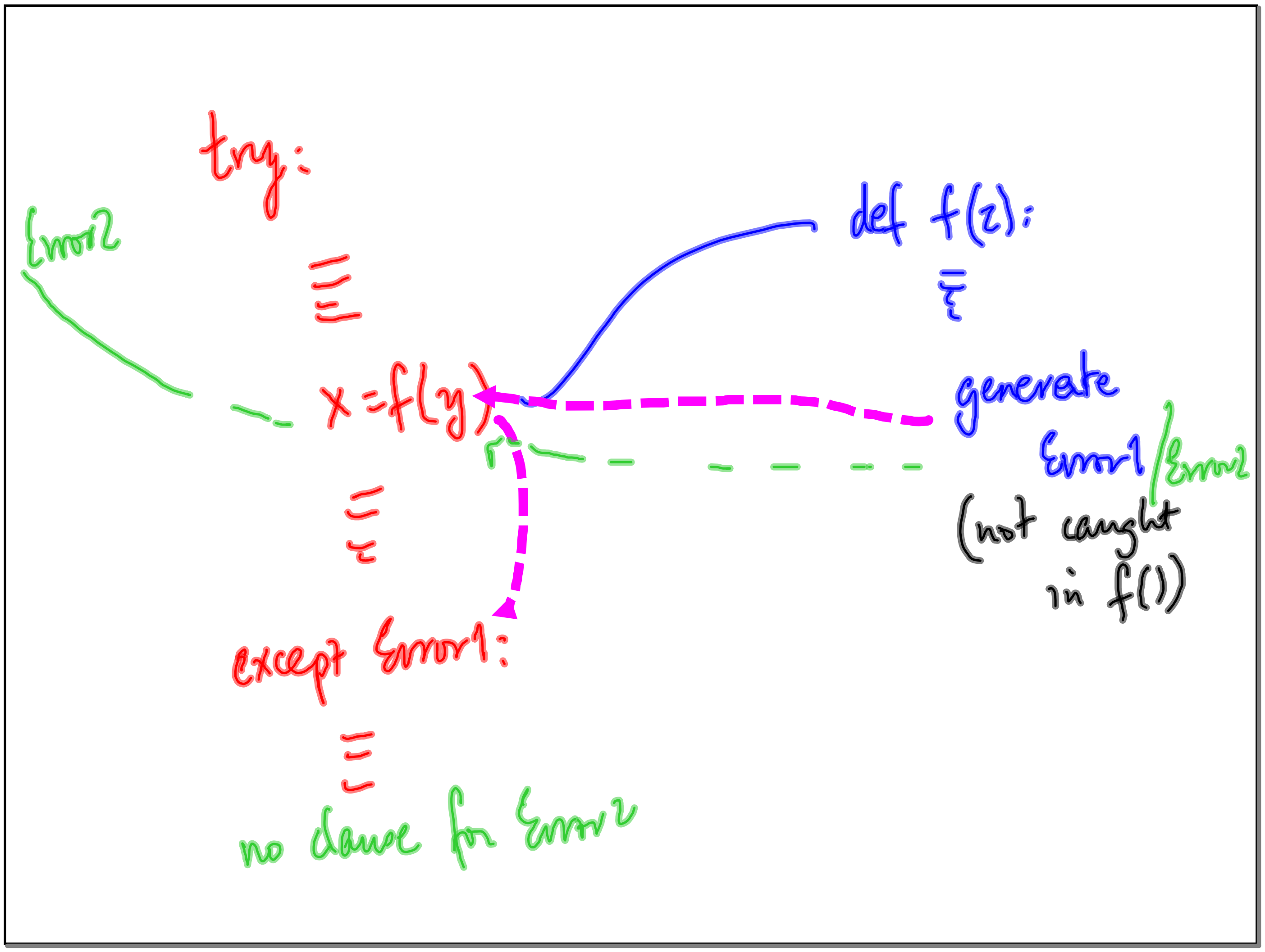
Catching errors in Python

```
try:
    ≡
except KeyError:
    ≡
except IndexError, NameError:
    ≡
```

```
try:
    ≡
except ErrorName1:
    ≡
except ErrorName2, ErrorName3:
    ≡
    ⋮
else:
    ≡   } If try ends normally, no error
finally:
    ≡   } Always executes, error or no error
```

optional




try:
Error2
x = f(y)
except Error1:
no clause for Error2
def f(z):
generate
Error1/Error2
(not caught
in f())

Back to our problem

```
try:
    score[name] = score[name] + newscore
except KeyError:
    score[name] = newscore
```

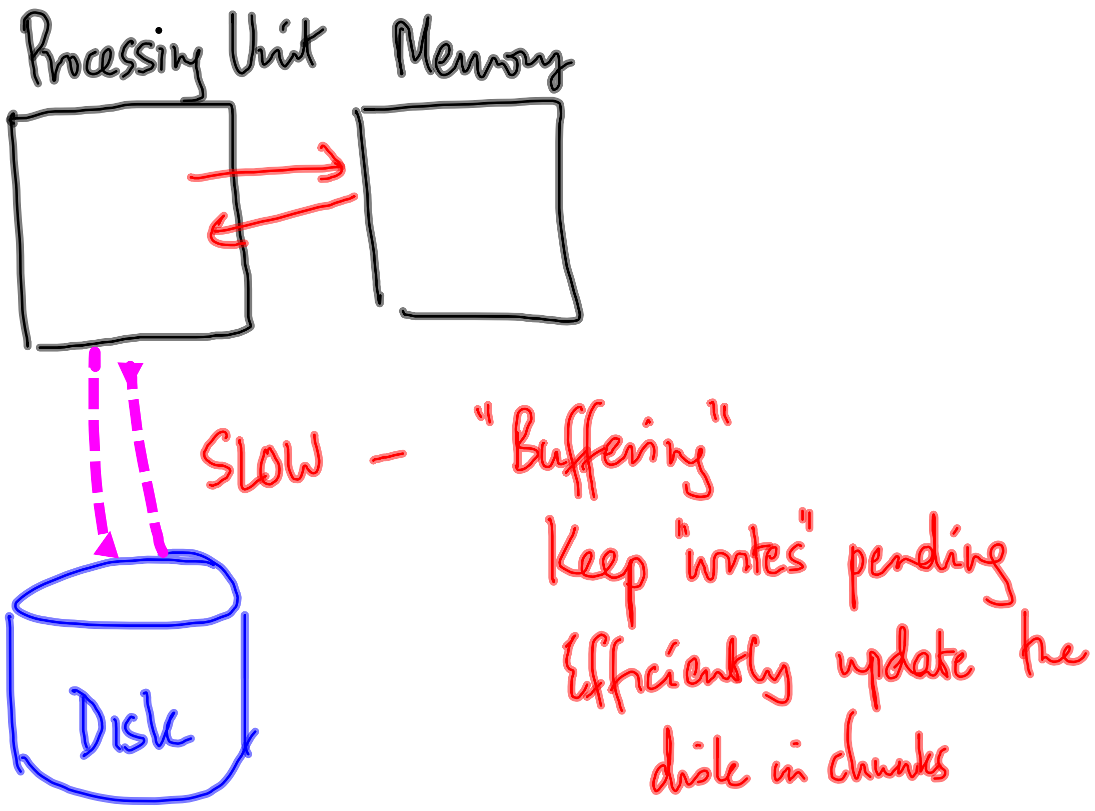
Input/Output

From keyboard : x = input()

To Screen : print(..)

More generally

Read & write data from/to files on disk


Processing Unit
Memory
Disk
SLOW - "Buffering"
Keep "writes" pending
Efficiently update the disk in chunks

$$\begin{tikzcd}\text{Programe} \arrow[r, bend right, dashed] & \boxed{\text{Buffer}} \arrow[l, bend right, dashed] \arrow[r, bend right, dashed] & \text{File} \arrow[l, bend right, dashed]\end{tikzcd}$$

Don't read/write files directly

2 step process

"open" file : connecting the file to a buffer

operate on buffer

"close" file : all pending updates are "flushed"

Opening a file creates "file handle"

```
fh = open('filename', 'mode')
```

name of file handle i.e. what we will read/write	name of file	r	read
		w	write
		a	append

```
:
fh.close()     # close file
```

Next time:

Reading & writing with file handles