Programming and Data Structures with Python

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

11 January, 2021

Breaking out of a loop

Finding the first position of `x` in `l`

Breaking out of a loop

Finding the first position of `x` in `l`

- Won't stop with the first position

```
def findpos(l,v):
    # Return first position of v in l
    # Return -1 if v not in l
    (pos,i) = (-1,0)
    while i < len(l):
        if l[i] == v:
            pos = i
        i = i+1
    # If pos not reset in loop, pos is -1
    return(pos)
```

Breaking out of a loop

Finding the first position of `x` in `l`

- Won't stop with the first position
- Add a flag `found`

```
def findpos2(l,v):
    # Return first position of v in l
    # Return -1 if v not in l
    (found,i) = (False,0)
    while i < len(l):
        if l[i] == v and not found:
            (found,pos) = (True,i)
        i = i+1
    if not found:
        pos = -1
    return(pos)
```

Breaking out of a loop

Finding the first position of `x` in `l`

- Won't stop with the first position
- Add a flag `found`
- Stop searching when `x` is seen

```
def findpos3(l,v):
    # Return first position of v in l
    # Return -1 if v not in l
    (found,i) = (False,0)
    while not(found) and i < len(l):
      # Changed while condition
      if l[i] == v and not found:
            (found,pos) = (True,i)
        i = i+1
    if not found:
        pos = -1
    return(pos)
```

redundant

Breaking out of a loop

Finding the first position of `x` in `l`

- Won't stop with the first position
- Add a flag `found`
- Stop searching when `x` is seen
- Instead, terminate loop, `break`

```
def findpos4(l,v):
    # Return first position of v in l
    # Return -1 if v not in l
    (pos,i) = (-1,0)
    while i < len(l):
        if l[i] == v:
            pos = i
            break
        i = i+1
    # If pos not reset, pos is -1
    return(pos)
```

Breaking out of a loop

Finding the first position of `x` in `l`

- Won't stop with the first position
- Add a flag `found`
- Stop searching when `x` is seen
- Instead, terminate loop, `break`
- Using `for` instead of `while`

```
def findpos5(l,v):
    (pos,i) = (-1,0)
    for x in l:
        if x == v:
            pos = i
            break
        i = i+1
    # If pos not reset in loop, pos is -1
    return(pos)
```

for, while

Breaking out of a loop

Finding the first position of `x` in `l`

- Won't stop with the first position
- Add a flag `found`
- Stop searching when `x` is seen
- Instead, terminate loop, `break`
- Using `for` instead of `while`
- A better version with `for`

```
def findpos6(l,v):
    pos = -1
    for i in range(len(l)):
        if l[i] == v:
            pos = i
            break
    # If pos not reset in loop, pos is -1
    return(pos)
```

How to tell if loop ended via break?

Breaking out of a loop

Finding the first position of `x` in `l`

- Won't stop with the first position
- Add a flag `found`
- Stop searching when `x` is seen
- Instead, terminate loop, `break`
- Using `for` instead of `while`
- A better version with `for`
- `else:` — executed with normal termination

```
def findpos7(l,v):
    for i in range(len(l)):
        if l[i] == v:
            pos = i
            break
    else:
        # No break, v not in l
        pos = -1
    return(pos)
```

executes if loop ends normally

Initialising names

- A name cannot be used before it is assigned a value

x is not assigned v

```
y = x + 1 # Error if x is unassigned
```

- May forget this for lists where update is implicit

```
l.append(v)
```

- Python needs to know that `l` is a list

Initialising names …

```
def factors(n):

  for i in range(1,n+1):
    if n%i == 0:
      flist.append(i)

  return(flist)
```

flist = flist+[i]

Initialising names ...

```
def factors(n):

  flist = []

  for i in range(1,n+1):
    if n%i == 0:
      flist.append(i)

  return(flist)
```

Generalizing lists

$l(0) = 13$

- `l = [13, 46, 0, 25, 72]`
  (0 1 2 3 4)
- View `l` as a function, associating values to positions
  - `l : {0,1,..,4} → integers`
  - `l(0) = 13, l(4) = 72`
- `0,1,..,4` are keys
- `l[0],l[1],..,l[4]` are corresponding values

Dictionaries

Collection

- Allow keys other than `range(0,n)`
- Key could be a string

```
test1["Dhawan"] = 84
test1["Pujara"] = 16
test1["Kohli"] = 200
```

test1["Kohli"] = 400

- Python dictionary
  - Any immutable value can be a key
  - Can update dictionaries in place —mutable, like lists

k1 → v1

k2 → v2

⋮

kn → vn

Tuples - immutable

Lists - mutable

Marks of students

- identify student (roll number)
- course (course code)

marks [(roll, code)] = m

× [roll, code] - mutable value cannot be used as key

Dictionaries

- Empty dictionary is `{}`, not `[]`
  - Initialization: `test1 = {}`
  - Note: `test1 = []` is empty list, `test1 = ()` is empty tuple
- Keys can be any immutable values
  - `int, float, bool, string, tuple`
  - But not lists, or dictionaries

[[0,1],[2,3]]

Dictionaries

k1 → v1
k2 → v2
⋮
kn → vn

- Can nest dictionaries

```
score["Test1"]["Dhawan"] = 84
score["Test1"]["Kohli"] = 200
score["Test2"]["Dhawan"] = 27
```

- Directly assign values to a dictionary

```
score = {"Dhawan":84, "Kohli":200}
score = {"Test1":{"Dhawan":84,
  "Kohli":200}, "Test2":{"Dhawan":50}}
```

score["Dhawan"] = [84, 100]

Operating on dictionaries

- `d.keys()` returns sequence of keys of dictionary `d`

```
for k in d.keys():
  # Process d[k]
```

list(d.keys()) $k_1, k_2, \ldots k_n$

$\downarrow$

$d[k_1], \ldots, d[k_n]$

- `d.keys()` is not in any predictable order

```
for k in sorted(d.keys()):
  # Process d[k]
```

- `sorted(l)` returns sorted copy of `l`, `l.sort()` sorts `l` in place
- `d.keys()` is not a list —use `list(d.keys())`

Operating on dictionaries

$\boxed{x \in L}$

- Similarly, `d.values()` is sequence of values in `d`

```
total = 0
for s in test1.values():
  total = total + test1
```

x in l

any sequence

- Test for key using `in`, like list membership

```
for n in ["Dhawan","Kohli"]:
  total[n] = 0
  for match in score.keys():
    if n in score[match].keys():
      total[n] = total[n] + score[match][n]
```

Dictionaries vs lists

- Assigning to an unknown key inserts an entry

```
d = {}
d[0] = 7  # No problem, d == {0:7}
```

- … unlike a list

```
l = []
l[0] = 7  # IndexError!
```