LEARN CODING

ale66

github.com/ale66/learn-coding

LOOPS

THE `While` INSTRUCTION

We saw `for` to *iterate* over a sequence

A block of code is executed as many times as the number of elements of the sequence

```
1  writers = ['Hemingway', 'Dickens', 'King']
2
3  for w in writers:
4    print(w)
```

With the while loop we execute the code block below only as long as the given expression is true

```
1  i = 1
2
3  while i < 6:
4    print(i)
5    i += 1
```

increment is not automatic

the expression is evaluated *before* executing the block

Visualise the example above on pythontutor.com:

which values will `i` take up? Which values will we see on the screen?

`break` can stop the loop even if the while condition is still true:

```
1  i = 1
2
3  while i < 6:
4
5    if i == 3:
6      break
7
8    print(i)
9    i += 1
```

INDEFINITE LOOPS

Would execute forever, or until a break condition is met

Example: the command-line-interface (CLI) terminal

```
1  while True:
2
3    command = input(''Please enter a command, type 'exit' to stop.'')
4
5    if command == 'exit':
6      break
7
8    else:
9      # implement command execution here ...
```

What is this code doing?

```
1  n = 5
2
3  while n > 0 :
4
5        print(n)
6
7        n = n-1   # or n -= 1
8
9  print('Blastoff!')
```

QUIZ: WHAT'S WRONG WITH THIS LOOP?

```
1  n = 10
2
3  while n > 0 :
4        print('Hi')
5
6  print('End')
```

Execution won't end as n is never decreased

INDEFINITE LOOPS

- some `while` loops are called *indefinite* because they keep going until a logical condition becomes false
- but will it ever happen?
- codes seen so far are easy to examine to see whether they will terminate or become *infinite*
- termination analysis is about checking, *before running it*, that our code will always reach the final nstruction and control will be back to the operating system
- this activity cannot be automated

`for` VS. `while`

Use `for`

- to automate work over collections, here represented by iterables
- to examine all elements (in principle)

Use `while`

- to search for a specific element: it stops as soon as it's found
- to act *conditionally* and stop as soon as the condition is not true anymore

while IN LISTS

```
my_list = [1, 3, 7, 9]

count = 0 # will work as index

while(True):
      print(my_list[count])

      if (count == len(my_list) - 1):
          print("end")
          break

      count += 1
```

```
for n in my_list:
        print(n)
```