Looping with for

The INFDEV Team @ HR

Hogeschool Rotterdam
Rotterdam, Netherlands

Introduction

Lecture topics

- the (lack of) limitations of `while` loops
- `for` statements and their semantics
- `for` as a *limited* form of `while`

`while` loops

Potential issues

- While loops specify unbounded iteration
- This means that the number of iterations is not necessarily easy to specify
- For example
  - Virtual machines
  - User-driven loops
  - Servers
  - Operating systems
  - ...

Unbounded loop example

```
n,m = input("Let's␣have␣two␣numbers")
cnt = 1
while n > m:
    cnt = cnt + 1
    n = n / m
print("Result␣is␣%d" % cnt)
```

What does this code do?

How many steps does it take?

Unbounded loop example

```
quit = False
while not quit:
  action = raw_input("Should␣I␣quit?")
  if (action == "Yes") | (action == "yes"):
    quit = True
  else:
    print("You␣are␣not␣a␣quitter.")
```

What does this code do?

How many steps does it take?

Unbounded loop example

```
y = 10.0
vy = 0.0
dt = 0.05
while (abs(vy) > 0.9) | (y > 0.2):
  new_y = y + vy * dt
  if new_y <= 0.1:
    vy = -vy * 0.7
  else:
    vy -= 9.8 * dt
    y = new_y
  cls()
  screen = ""
  for j in range(0,20):
    for i in range(0,20):
      if (i == 10) & (j == 19 - int(y)):
        screen += "O"
      elif j == 19:
        screen += "-"
      else:
        screen += "␣"
    screen += "\n"
  print(screen)
  sleep(0.01)
```

What does this code do?

How many steps does it take?

while loops

Potential issues

- `while` loops are very powerful
- with great power comes...

while loops

Potential issues

- while loops are very powerful
- with great power comes...
- ...greater chance of bugs

Unbounded loop example

```
y = 10.0
vy = 0.0
dt = 0.05
while (abs(vy) > 0.9) | (y > 0.1):
  new_y = y + vy * dt
  if new_y <= 0.1:
    vy = -vy * 0.7
  else:
    vy -= 9.8 * dt
    y = new_y
  cls()
  screen = ""
  for j in range(0,20):
    for i in range(0,20):
      if (i == 10) & (j == 19 - int(y)):
        screen += "O"
      elif j == 19:
        screen += "-"
      else:
        screen += "␣"
    screen += "\n"
  print(screen)
  sleep(0.01)
```

Does this loop terminate? (This is not the same code as before!)

HOGESCHOOL ROTTERDAM

Unbounded loop example

```
y = 10.0
vy = 0.0
dt = 0.05
while (abs(vy) > 0.9) | (y > 0.1):
  new_y = y + vy * dt
  if new_y <= 0.1:
    vy = -vy * 0.7
  else:
    vy -= 9.8 * dt
    y = new_y
  cls()
  screen = ""
  for j in range(0,20):
    for i in range(0,20):
      if (i == 10) & (j == 19 - int(y)):
        screen += "O"
      elif j == 19:
        screen += "-"
      else:
        screen += "␣"
    screen += "\n"
  print(screen)
  sleep(0.01)
```

Does this loop terminate? (This is not the same code as before!)

No. The condition has changed to `y >␣0.1`.

Unbounded loop example

```
y = 10.0
vy = 0.0
dt = 0.1
while (abs(vy) > 0.9) | (y > 0.1):
  new_y = y + vy * dt
  if new_y <= 0.1:
    vy = -vy * 0.8
  else:
    vy -= 9.8 * dt
    y = new_y
  cls()
  screen = ""
  for j in range(0,20):
    for i in range(0,20):
      if (i == 10) & (j == 19 - int(y)):
        screen += "O"
      elif j == 19:
        screen += "-"
      else:
        screen += "␣"
    screen += "\n"
  print(screen)
  sleep(0.01)
```

Does this loop terminate? (This is not the same code as before!)

HOGESCHOOL ROTTERDAM

Unbounded loop example

```
y = 10.0
vy = 0.0
dt = 0.1
while (abs(vy) > 0.9) | (y > 0.1):
  new_y = y + vy * dt
  if new_y <= 0.1:
    vy = -vy * 0.8
  else:
    vy -= 9.8 * dt
    y = new_y
  cls()
  screen = ""
  for j in range(0,20):
    for i in range(0,20):
      if (i == 10) & (j == 19 - int(y)):
        screen += "O"
      elif j == 19:
        screen += "-"
      else:
        screen += "␣"
    screen += "\n"
  print(screen)
  sleep(0.01)
```

Does this loop terminate? (This is not the same code as before!)

No. `dt = 0.1` and `vy = -vy * 0.8`.

Correctly encoding intentions

Why is `while` not enough

- The expressive power of `while` is not always needed
- Sometimes we want something simpler, and less dangerous
- For example, consider:
  - For each *hostile alien*
  - Do *attack it*

Correctly encoding intentions

Why is `while` not enough

- A loop such as:
  - For each *hostile alien*
  - Do *attack it*
- Is predictable
- Performs a fixed number of steps (one per hostile alien)
- Will certainly terminate

Correctly encoding intentions

Why is `while` not enough

- In general, we wish to always correctly encode our intention of repeating code N times
- The code must precisely fit our intentions, like a tailored italian suit
  - Code should not be too complicated
  - Code should not be too simple

Correctly encoding intentions

Code that is too complicated?

- A `while` loop where we need to perform `N` steps
- There are many subtle ways to break the code

Correctly encoding intentions

Code that is too complicated?

- Classes, objects, and inheritance everywhere
- To know which code is actually run to say `Hello world!` you need to read twelve files

Correctly encoding intentions

Code that is too complicated?

- Events, lambda's, higher-order combinators everywhere
- To know what the program does you need two doctorates (CompSci and Maths)
  - Plus internal access to the sliced brain of the original programmer

Correctly encoding intentions

Code that is too simple?

- No handling of error cases
- Ignoring hard circumstances

Correctly encoding intentions

Code that is too simple?

- No handling of error cases
- Ignoring hard circumstances
- Not implementing all features correctly
    - Showing progress off
    - Building impressive but pointless demo's

Iterating with `for`

Code that is too simple?

- Python, and many other modern languages, offer explicit constructs for bounded repetition
  - We specify precisely the number of steps that need to be performed
  - The language takes care of performing the right number of steps
  - The construct is much harder to break[a] than a `while`-loop
- These constructs are called for-loops

[a] Running forever

Iterating with for

Syntax of for

- Number of repetitions (a `range` iterator)
- That stores the index of the current repetition (a variable)
- Body of the loop that is repeated at every iteration (a block of code)

Syntax of `for`

```
for VARIABLE in range(END):
  BODY
```

- `VARIABLE` is any valid variable name that becomes useable within the `BODY`; will range from `0` to `END-1`
- `END` is any positive number; the body will be repeated `END-1` times
- `BODY` is a series of statements

Iterating with for

Examples of for

- Number of repetitions (a range iterator)
- That stores the index of the current repetition (a variable)
- Body of the loop that is repeated at every iteration (a block of code)

This is it!

The best of luck, and thanks for the attention!