Conditionals

The INFDEV Team @ HR

Hogeschool Rotterdam
Rotterdam, Netherlands

Introduction

Lecture topics

- Making choices
- `if-then-else` statements
- Reasoning about `if-then-else`

Making choices

Making choices

- Often need to *make a choice*
- Based on some *condition*, we do *something* rather than *something else*

Making choices

Making choices

- If *the sun is shining*
- Then *take a walk*
- Otherwise *go to work*

Making choices

Making choices

- If *the engine is too warm and the RPM's are high enough*
- Then *reduce the RPM*
- Otherwise *do nothing*

Making choices

Making choices

- Of course conditions like this can be combined
- This means that we can *cascade* decisions
- This is the building block of *intelligent decisions* in our programs

Making choices

Making choices

- If *the engine is too warm*
- Then
  - If *the RPM's are high enough*
  - Then *reduce the RPM*
  - Otherwise *light up the temperature lamp*
- Otherwise *do nothing*

Making decisions in Python

`if-then-else`

- Python offers built-in facilities for decision-making
- `if-then-else` statement
- We can make decisions about which block of code is executed

Making decisions in Python

`if-then-else`

- The general form is `if CONDITION: THEN-BLOCK else ELSE-BLOCK` (if_{CTE})
- If the condition is true, then we jump to the beginning of `THEN-BLOCK`, otherwise we jump to the beginning of `ELSE-BLOCK`

$$\begin{cases} (PC, S) \overset{if_{CTE}}{\rightarrow} (firstLine(T), S) & when \quad (PC, S) \overset{C}{\rightarrow} TRUE \\ (PC, S) \overset{if_{CTE}}{\rightarrow} (firstLine(E), S) & when \quad (PC, S) \overset{C}{\rightarrow} FALSE \end{cases}$$

Making decisions in Python

if-then-else

- Python is *indentation*-based
- White-spaces go at the beginning of some lines
- A more indented line is *within* a less indented line above

Making decisions in Python

`if-then-else`

- Indentation specifies where the `then-block` and the `else-block` begin and end
- The general form of an `if-then-else` is thus:
  - `if COND:`
  - `newline`
  - **indentation**
  - `code of then`
  - **de-indentation**
  - `else:`
  - `newline`
  - **indentation**
  - `code of else`
  - **de-indentation**

A correct example

```
if temp > 350.0:
  if throttle > 2500:
    throttle = throttle - 1500
  else:
    warning = True
else:
  print("everything␣is␣OK")
```

An incorrect example

```
if temp > 350.0:
if throttle > 2500:
throttle = throttle - 1500
else:
warning = True
else:
print("everything␣is␣OK")
```

Making decisions in Python

if-then-else

- if-then-else statements eventually terminate
- after the then (or else) block is finished, we jump to the first line right after the whole if-then-else

After an if-then-else

```
if temp > 350.0:
  if throttle > 2500:
    throttle = throttle - 1500
  else:
    warning = True
else:
  print("everything␣is␣OK")
print(throttle, temp, warning)
```

After an `if-then-else`?

Without indentation, this:

```
if temp > 350.0:
if throttle > 2500:
throttle = throttle - 1500
else:
warning = True
else:
print("everything␣is␣OK")
print(throttle, temp, warning)
```

would be indistinguishable from both:

```
if temp > 350.0:
  if throttle > 2500:
    throttle = throttle - 1500
  else:
    warning = True
else:
  print("everything␣is␣OK")
  print(throttle, temp, warning)
```

```
if temp > 350.0:
  if throttle > 2500:
    throttle = throttle - 1500
  else:
    warning = True
else:
  print("everything␣is␣OK")
print(throttle, temp, warning)
```

Reasoning about `if-then-else`

`if-then-else`

- `if-then-else` effectively forks the code
- Until run-time, we are not really sure what path the code will take

Example if's

```
x = 0
y = 0
z = 0
op = "none"
input = sys.stdin.readline()
if input == "*\n":
  x = int(sys.stdin.readline())
  y = int(sys.stdin.readline())
  op = "*"
else:
  if input == "+\n":
    x = int(sys.stdin.readline())
    y = int(sys.stdin.readline())
    op = "+"
  else:
    x = int(sys.stdin.readline())
    y = 2
    op = "*"
```

```
if op == "+":
  z = x + y
else:
  if op == "*":
    z = x * y
  else:
    raise
print(str(x) + "␣" + op + "␣" +
        str(y) + "␣is␣" + str(z))
```

Example if's

Which path will be taken?

```
x = 0
y = 0
z = 0
op = "none"
input = sys.stdin.readline()
if input == "*\n":
  x = int(sys.stdin.readline())
  y = int(sys.stdin.readline())
  op = "*"
else:
  if input == "+\n":
    x = int(sys.stdin.readline())
    y = int(sys.stdin.readline())
    op = "+"
  else:
    x = int(sys.stdin.readline())
    y = 2
    op = "*"
```

Example if's

Which path will be taken?

```
x = 0
y = 0
z = 0
op = "none"
input = sys.stdin.readline()
if input == "*\n":
  x = int(sys.stdin.readline())
  y = int(sys.stdin.readline())
  op = "*"
else:
  if input == "+\n":
    x = int(sys.stdin.readline())
    y = int(sys.stdin.readline())
    op = "+"
  else:
    x = int(sys.stdin.readline())
    y = 2
    op = "*"
```

We do not know!

Example `if`'s

What values will `x`, `y`, `op`, `input` have?

```
x = 0
y = 0
z = 0
op = "none"
input = sys.stdin.readline()
if input == "*\n":
  x = int(sys.stdin.readline())
  y = int(sys.stdin.readline())
  op = "*"
else:
  if input == "+\n":
    x = int(sys.stdin.readline())
    y = int(sys.stdin.readline())
    op = "+"
  else:
    x = int(sys.stdin.readline())
    y = 2
    op = "*"
```

Example if's

What values will x, y, op, input have?

```
x = 0
y = 0
z = 0
op = "none"
input = sys.stdin.readline()
if input == "*\n":
  x = int(sys.stdin.readline())
  y = int(sys.stdin.readline())
  op = "*"
else:
  if input == "+\n":
    x = int(sys.stdin.readline())
    y = int(sys.stdin.readline())
    op = "+"
  else:
    x = int(sys.stdin.readline())
    y = 2
    op = "*"
```

We do not know!

Reasoning about if-then-else

if-then-else

- The paths are influenced by the value of the input variable
    - One path for "*\n"
    - Another for "+\n"
    - Another for all other possible values
- We analyze our code based on all possible outcomes

Example if's

x	y	z	op	input
0	0	0	"none"	"*\n"

```
if input == "*\n":
  x = int(sys.stdin.readline())
  y = int(sys.stdin.readline())
  op = "*"
else:
  ...
```

Example if's

x	y	z	op	input
0	0	0	"none"	"*\n"

```
if input == "*\n":
  x = int(sys.stdin.readline())
  y = int(sys.stdin.readline())
  op = "*"
else:
  ...
```

x	y	z	op	input
in2	in3	0	"*"	"*\n"

Example if's

x	y	z	op	input
0	0	0	"none"	"+\n"

```
if input == "*\n":
  ...
else:
  if input == "+\n":
    x = int(sys.stdin.readline())
    y = int(sys.stdin.readline())
    op = "+"
  else:
    ...
```

Example if's

x	y	z	op	input
0	0	0	"none"	"+\n"

```
if input == "*\n":
  ...
else:
  if input == "+\n":
    x = int(sys.stdin.readline())
    y = int(sys.stdin.readline())
    op = "+"
  else:
    ...
```

x	y	z	op	input
in2	in3	0	"+"	"+\n"

Example if's

x	y	z	op	input
0	0	0	"none"	"anything else"

```
if input == "*\n":
  ...
else:
  if input == "+\n":
    ...
  else:
    x = int(sys.stdin.readline())
    y = 2
    op = "*"
```

Example if's

x	y	z	op	input
0	0	0	"none"	"anything else"

```
if input == "*\n":
  ...
else:
  if input == "+\n":
    ...
  else:
    x = int(sys.stdin.readline())
    y = 2
    op = "*"
```

x	y	z	op	input
in2	2	0	”*”	”anything else”

Example `if`'s

We can now merge the various possible outcomes (ignoring `input` as we do not use it anymore):

x	y	z	op	input
in2	in3	0	"*"	"*\n"
in2	in3	0	"+"	"+\n"
in2	2	0	"*"	"anything else"

Example `if`'s

We can now merge the various possible outcomes (ignoring `input` as we do not use it anymore):

x	y	z	op	input
in2	in3	0	"*"	"*\n"
in2	in3	0	"+"	"+\n"
in2	2	0	"*"	"anything else"

x	y	z	op
in2	in3 $\vee$ 2	0	"*" $\vee$ "+"

Example if's

x	y	z	op
in2	in3 ∨ 2	0	"*" ∨ "+"

```
if op == "+":
  z = x + y
else:
  if op == "*":
    z = x * y
  else:
    raise
```

Example if's

x	y	z	op
in2	in3 ∨ 2	0	"*" ∨ "+"

```
if op == "+":
  z = x + y
else:
  if op == "*":
    z = x * y
  else:
    raise
```

x	y	z	op
in2	in3 ∨ 2	$in2+in3 \vee in2\times in3 \vee in2\times 2$	"*" ∨ "+"

Reasoning about `if-then-else`

Exponential explosion of potential control-paths

- `in2+in3` $\vee$ `in2`$\times$`in3` $\vee$ `in2`$\times$`2` is a long formula
- It is simply saying that there are three possible outcomes:
  - One outcome is `in2+in3`
  - One outcome is `in2`$\times$`in3`
  - One outcome is `in2`$\times$`2`

Reasoning about `if-then-else`

Exponential explosion of potential control-paths

- The more sequential conditionals, the more possible resulting execution paths
- But *how many?*

Sequential if's

How many if's?
How many execution paths?

```
if C1:
  A1
else:
  B1
```

Sequential if's

How many if's?
How many execution paths?

```
if C1:
  A1
else:
  B1
```

1 if
2 execution paths

Sequential if's

```
if C1:
  A1
else:
  B1

if C2:
  A2
else:
  B2
```

Sequential if's

```
if C1:
  A1
else:
  B1

if C2:
  A2
else:
  B2
```

2 `if's`

4 execution paths

Sequential if's

```
if C1:
  A1
else:
  B1

if C2:
  A2
else:
  B2

if C3:
  A3
else:
  B3
```

Sequential if's

```
if C1:
  A1
else:
  B1

if C2:
  A2
else:
  B2

if C3:
  A3
else:
  B3
```

3 if's

8 execution paths

Reasoning about `if-then-else`

Exponential explosion of potential control-paths

- In general, for `n` `if`'s
- 2^n possible execution paths

Reasoning about `if-then-else`

Exponential explosion of potential control-paths

- Each path can alter the state in a different way
- After an `if` with 8 possible paths
  - We have 8 possible resulting states
  - Variables can be one of possible 8 different values

Reasoning about `if-then-else`

Exponential explosion of potential control-paths

- The more `if`'s
- The more complex its conditions
- *The harder it is to reason about your program!*

Using `if`'s

Rules of thumb

- Logical, short condition
- Good: `(temp > 350 & throttle > 2500)`
- Bad: `(temp > 350 & throttle > 2500 & op == "+")`

Using `if`'s

Rules of thumb

- Few levels of nesting
- Good: between one and three
- Bad: beyond three

Using `if`'s

Rules of thumb

- Semantically connected `then` and `else`
- Good: both `then` and `else` perform similar operations on the same variables
- Bad: `then` and `else` perform unrelated operations or on different variables

A disastrous example

```
if (temp > 350 & throttle > 2500) | op == "+":
  if op == "+":
    z = x + y
  else:
    z = x * y
    throttle = throttle - 1000
else:
  if op == "*":
    z = x * y
```

What went wrong?

A disastrous example

```
if (temp > 350 & throttle > 2500) | op == "+":
  if op == "+":
    z = x + y
  else:
    z = x * y
    throttle = throttle - 1000
else:
  if op == "*":
    z = x * y
```

What went wrong?

- The condition is very hard to reason about
- The condition involves unrelated quantities
- The various `then`'s and `else`'s are partially unrelated
- There is repetition

Bringing order

```
if temp > 350 & throttle > 2500:
  throttle = throttle - 1000

if op == "+":
  z = x + y
else:
  z = x * y
```

What went right?

Bringing order

```
if temp > 350 & throttle > 2500:
  throttle = throttle - 1000

if op == "+":
  z = x + y
else:
  z = x * y
```

What went right?

- The conditions are simple to reason about
- The conditions are all tight (no unrelated variables)
- The various `then`'s and `else`'s are all strongly related
- Separate `if`'s for separate tasks
- There is no repetition

Using `if`'s

The value of reasoning

- **Always keep in mind:**
- You have the power to make your own life a living Hell...

Using if's

The value of reasoning

- **Always keep in mind:**
- You have the power to make your own life a living Hell...
- ...unless you reason first and then structure code logically

This is it!

The best of luck, and thanks for the attention!