ECOR 1041
Computation and Programming

Control Flow: Conditional Statements

References

- *Practical Programming*, 3rd ed.
  - Chapter 5, pp. 77, 80 - 82, 86 – 94; This is:
    - Chapter introduction
    - In Section “A Boolean Type”, the subsection *Relational Operators*
    - Sections “Choosing Which Statements to Execute”, “Nested If Statements”, and “Remembering Results of a Boolean Expression Evaluation”

Lecture Objectives

- Introduce and apply these conditional statements: `if`, `if-else` and `if-elif-else`

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - Type `bool`, Boolean values
  - Relational operators
  - Control flow, conditional statements
  - `if`, `else` and `elif` keywords

Learning Outcomes

- Be able to design and implement functions that use `if`, `if-else` and `if-elif-else` statements

Type `bool`

- Provides the set of Boolean values (`True`, `False`) and operators for those values
- Note the spelling of the literal values: first letter is uppercase, remaining letters are lowercase

Relational Operators

- Compare two values, produce `True` or `False`

Operator	Operation	Example
>	Greater than	`20 > 10 ⇒ True`
<	Less than	`20 < 10 ⇒ False`
>=	Greater than or equal to	`5 >= 5 ⇒ True`
<=	Less than or equal to	`10 <= 5 ⇒ False`
==	Equal to	`42 == 42 ⇒ True`
!=	Not equal to	`10 != 20 ⇒ True`

Carleton University

Control Flow (Flow of Control)

- *Control flow*: the order in which statements in a program (written in an imperative programming language, like Python) are executed
- *Control flow statements* choose between two or more paths of execution through the code
- In this course, the control flow statements we will use are *conditional* statements and *loop (repetition)* statements

Conditional Statements

- Conditional statements perform different computations depending on whether a Boolean condition evaluates to `True` or `False`
- In this lecture, we will introduce Python's `if`, `if-else` and `if-elif-else` statements

Example: Calculating Absolute Values

- The absolute value of a real number x is denoted by $|x|$ and is defined as:

  $|x| = x * \text{sgn}(x)$, where

  $$\text{sgn}(x) = \begin{cases} -1, & \text{for } x < 0 \\ 0, & \text{for } x = 0 \\ 1, & \text{for } x > 0 \end{cases}$$

- So, $|x| = -x$, for $x \leq 0$
  $\quad\quad = x$, for $x \geq 0$

Example: Calculating Absolute Values

- When implementing |*x*| as a Python function, we will use a conditional statement that specifies what the function will do (return *x* or -*x*) based on a condition (is *x* positive or negative or 0?)

`if` statement

- Syntax:

```
if condition:
    block1
```

  - *`block1`* must be indented to denote that it belongs to the `if` statement
- Semantics: if *`condition`* evaluates to `True`, execute the statements in *`block1`*

`if-else` statement

- Syntax:

```
if condition:
    block1
else:
    block2
```

block1 and *block2* must be indented by the same number of spaces

- Semantics: if *condition* evaluates to `True`, execute the statements in *block1*, otherwise execute the statements in *block2*

Absolute Value (Version 1)

- Uses `if-else`, two `return` statements

```
def abs(x: float) -> float:
    if x >= 0:
        return x
    else:               # x < 0
        return -x
```

- Use Python Tutor to trace the control flow (link posted on Brightspace)

Absolute Value (Version 2)

- Uses `if` (no `else` clause)

```
def abs(x: float) -> float:
    if x >= 0:
        return x
    return -x    # x < 0
```

`return -x` is outside the `if` statement

- Use Python Tutor to trace the control flow
- While Version 1 works, Version 2 is more elegant and demonstrates we understand that an `else` is not required after a `return`

Absolute Value (Versions 3 and 4)

- Some programmers believe functions should have exactly one `return` statement, which must be the last statement in the function body
- Versions 3 and 4 follow this convention

Absolute Value (Version 3)

- Uses `if-else` and one `return` statement

```
def abs(x: float) -> float:
    if x >= 0:
        abs_x = x
    else:        # x < 0
        abs_x = -x
    return abs_x
```

Local variable `abs_x` refers to the value that will be returned

Carleton University

Absolute Value (Version 4)

- Uses `if` (no `else` clause) and one `return` statement

```
def abs(x: float) -> float:
    abs_x = x
    if x < 0:
        abs_x = -x
    return abs_x
```

Initially assumes $x \geq 0$, so assigns `x` to `abs_x`, then checks if $x < 0$

We could get the same behaviour without `abs_x`. How?

Absolute Value (Version 5 - Poor Style)

```
def abs(x: float) -> float:
    if x >= 0:
        return x
    if x < 0:
        return -x
```

- Conditions `x >= 0` and `x < 0` are mutually exclusive
- If `x >= 0` is `False`, `x < 0` must be `True`, so evaluating condition `x < 0` is redundant

Traffic Light Colours: Function Header

- Define a function that returns the next colour that will be displayed by a traffic light, given the current colour

```
def traffic_light(current: str) -> str:
```

- The function takes a character string (a value of type `str`) and returns a character string (we will learn more about `str`s soon!)
- Note that we are implementing North American traffic lights: red -> green -> yellow -> red (etc.)

`traffic_light`: Header and Docstring

```
def traffic_light(current: str) -> str:
    """Return the next colour that will be
    displayed by a traffic light, after
    the current colour.

    Precondition: current is "red", "green",
    or "yellow".
```

Carleton University

`traffic_light` Docstring Examples

```
>>> traffic_light("red")
"green"

>>> traffic_light("green")
"yellow"

>>> traffic_light("yellow")
"red"
"""
```

Carleton University

Traffic Light Colours (Version 1)

- Evaluating a single condition is not sufficient
  - There are three possible argument values, with a different result returned for each one
- To handle this, we can use multiple `if` statements

Traffic Light Colours (Version 1)

```
def traffic_light(current: str) -> str:
    if current == "red":
        next = "green"
    if current == "green":
        next = "yellow"
    if current == "yellow":
        next = "red"
    return next
```

Traffic Light Colours (Version 2)

- In Version 1, all three conditions are always evaluated, even though only one of the `next = colour` statements will be executed
- Rewrite the function to use *nested* `if` statements (an `if` statement inside an `if` statement)
- After `next` is assigned the next colour, no further conditions are evaluated

Traffic Light Colours (Version 2)

```
def traffic_light(current: str) -> str:
    if current == "red":
        next = "green"
    else:
        if current == "green":
            next = "yellow"
        else:     # current == "yellow"
            next = "red"
    return next
```

Traffic Light Colours (Version 3)

- When the block in an `else` clause is an `if` statement or an `if-else` statement, we can replace

```
    else:
        if condition:
            block
```

with

```
    elif condition:
        block
```

`if-elif` statement

- Syntax:

```
if condition1:
    block1
elif condition2:
    block2
```

- Semantics:
  - if *`condition1`* evaluates to `True`, execute *`block1`*
  - otherwise, if *`condition2`* evaluates to `True`, execute *`block2`*

Carleton University

`if-elif-else` statement

- Syntax:

```
if condition1:
    block1
elif condition2:
    block2

else:
    block3
```

`if-elif-else` statement

- Semantics:
    - if *`condition1`* evaluates to `True`, execute *`block1`*
    - otherwise, if *`condition2`* evaluates to `True`, execute *`block2`*
    - otherwise, execute *`block3`*

Traffic Light Colours (Version 3)

```
def traffic_light(current: str) -> str:
    if current == "red":
        next = "green"
    elif current == "green":
        next = "yellow"
    else:     # current == "yellow"
        next = "red"
    return next
```

How can we re-write this with multiple returns?

Recap of Learning Outcomes

Carleton University

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - Type `bool`, Boolean values
  - Relational operators
  - Control flow, conditional statements
  - `if`, `else` and `elif` keywords

Learning Outcomes

- Be able to design and implement functions that use `if`, `if-else` and `if-elif-else` statements