1811/2807/7001ICT
Programming Principles

School of Information and Communication Technology
Griffith University

Trimester 1, 2024

11 Selections With if

Programs can make decisions whether or not to execute statements or not with the `if` statement.

11.1 Control flow statements

As with any imperative languge, control flows through the program statements from top to bottom, except where special control flow statements alter that sequence with selections and loops.

Selections are structures where programs decide to execute some statements or not.

Loops are structures where programs repeat statements many times.

11.2 if

This template shows how to construct an `if` statement.

```
if condition:
   actions(s)
```

The important parts:

- The `if` keyword starts an `if` statement.
- The *`condition`* is an expression that may be evaluated to `True` or `False`.
- The *`condition`* is separated from the *`actions`* by a colon (`:`).

- The *actions* are statements that will be executed if and only if the *condition* is True.
- The *actions* are *indented*!

Example scripts:

```
# file: if1.py
# One action statement, version 1.

print("You win if you guess the magic number.")
n = int(input("Enter a number: "))
if n == 3:
   print("You win!")
```

In this example there is only one action statement it could also be written like this:

```
# file: if2.py
# One action statement, version 2.

print("You win if you guess the magic number.")
n = int(input("Enter a number: "))
if n == 3: print("You win!")
```

It is more compact, but it is *less readable*.

In this example there are multiple action statements, and a statement after the `if` statment.

```
# file: if3.py
# Multiple action statements.

print("You win if you guess the magic number.")
n = int(input("Enter a number: "))
if n == 3:
   print("You win!")
   print("It is the first odd prime. Yay!")
print("If you didn't win, better luck next time.")
```

The indenting is very important.

The indented statements are inside the `if` statement.

The last statement is not inside the `if` statment and will always be executed.

```
$ python3 if3.py
You win if you guess the magic number.
Enter a number: 7
If you didn't win, better luck next time.
$ python3 if3.py
You win if you guess the magic number.
Enter a number: 3
You win!
It is the first odd prime. Yay!
If you didn't win, better luck next time.
$
```

11.3 else

The `else` keyword extends the `if` statement with alternative actions to execute when the *`condition`* is `False`.

```
if condition:
   actions(s)
else:
   actions(s)
```

```
# file: ifElse1.py
# Actions and alternatives.

print("You win if you guess the magic number.")
n = int(input("Enter a number: "))
if n == 3:
   print("You win!")
   print("It is the first odd prime. Yay!")
else:
   print("Sorry.")
   print("Better luck next time.")
```

```
$ python3 ifElse1.py
You win if you guess the magic number.
Enter a number: 3
You win!
It is the first odd prime. Yay!
$ python3 ifElse1.py
You win if you guess the magic number.
Enter a number: 7
Sorry.
Better luck next time.
$
```

11.4 elif

When we want to choose between more than two alternative *actions* using more than one *condition* we insert `elif` (short for *else if*) clauses.

```
if condition:
   actions(s)
elif condition:
   actions(s)
⋮
else:
   actions(s)
```

Example:

```
# file: ifElifElse1.py
# More actions and alternatives.

print("You win if you guess the magic number.")
n = int(input("Enter a number: "))
if n == 3:
   print("You win!")
   print("It is the first odd prime. Yay!")
elif n == 7:
   print("Some people think so, but not me.")
   print("Try again.")
else:
   print("Sorry.")
   print("Better luck next time.")
```

11.5 Indenting advice

Most modern programming languages use indenting to help the human reader understand the structure of a program.

Most languages' compilers or interpreters ignore the indenting. It is only for human readers.

Such languages need a lot more punctuation symbols, such as braces and semicolons, to define the structure.

Python is one of a few that defines the structure with mandatory indenting.

11.5.1 How much to indent?

This is a matter of taste.

Most people indent their code in steps of 3 or 4 space-widths.

Be consistent!

11.5.2 Spaces or tabs?

Many programmers indent their code with the tab character. Many others type their spaces manually.

The tab character is a single character that is expanded to multiple spaces when the text is displayed.

On plain, old dumb terminals and teletypes the space value of a tab was 8.

Modern text editors let you define that number, but that number is a property of the editor, not your script.

If you use tabs, how your code looks will depend on the editor or app that is displaying it.

The Python interpreter does not always see what you do.

This is a screen shot of my text editor, with tabs set to 3 spaces.

tabs1.py

~/Dropbox/.../.../.../11/samples-11/tabs... (functions)

```
1  # file: tabs1.py
2  # Fun with tabs.
3
4  print("You win if you guess the magic number.")
5  n = int(input("Enter a number: "))
6  if n == 3:
7     print("You win!")
8     print("It is the first odd prime. Yay!")
9  print("If you didn't win, better luck next time.")
10
```

L: 10 C: 1 Python Western (Mac OS Roman) Unix (LF)

Looks good, but the interpreter hates it.

```
$ python3 tabs1.py
  File "tabs1.py", line 8
    print("It is the first odd prime. Yay!")
                                            ^
IndentationError: unindent does not match any outer
indentation level
$
```

My editor can display the invisible characters, spaces as dots and tabs as Δ.

tabs1.py

~/Dropbox/.../.../.../11/samples-11/tabs... (functions)

```
1  #·file:·tabs1.py¬
2  #·Fun·with·tabs.¬
3  ¬
4  print("You·win·if·you·guess·the·magic·number.")¬
5  n·=·int(input("Enter·a·number:·"))¬
6  if·n·==·3:¬
7  Δ  print("You·win!")¬
8  ···print("It·is·the·first·odd·prime.·Yay!")¬
9  print("If·you·didn't·win,·better·luck·next·time.")¬
10
```

L: 10 C: 1 Python Western (Mac OS Roman) Unix (LF)

Python thinks tabs are worth the traditional, old-fashioned 8 spaces.

So this it what it sees.

tabs1.py

~/Dropbox/.../.../.../11/samples-11/tabs... (functions)

```
1  #·file:·tabs1.py¬
2  #·Fun·with·tabs.¬
3  ¬
4  print("You·win·if·you·guess·the·magic·number.")¬
5  n·=·int(input("Enter·a·number:·"))¬
6  if·n·==·3:¬
7  Δ       print("You·win!")¬
8  ···print("It·is·the·first·odd·prime.·Yay!")¬
9  print("If·you·didn't·win,·better·luck·next·time.")¬
10
```

L: 4 C: 47 Python Western (Mac OS Roman) Unix (LF)

Oops.

11.6 Nesting

The actions inside an `if` statement may be simple function calls or assignment statements, but they may also be structured control statements.

So we can put `if` statements inside `if` statements.

```
# file: ifIf1.py
# Nested ifs.
```

```
print("You win if you guess the magic number.")
n = int(input("Enter a number: "))
if n == 3:
   print("You win!")
   print("It is the first odd prime. Yay!")
else:
   if n == 7:
      print("Some people think so, but not me.")
   print("Better luck next time.")
   if n < 3:
      print("Try higher.")
   else:
      print("Try lower.")
```

Exercise: As a class, predict what this program will print for 1, 3, 7, and 9.

Section summary and further reading

This section covered:

- how to write selections using `if`, `else`, and `elif`;
- how to indent; and
- nested `if` statements.

For this section you should also read:

- Developers Who Use Spaces Make More Money Than Those Who Use Tabs (Hah!).