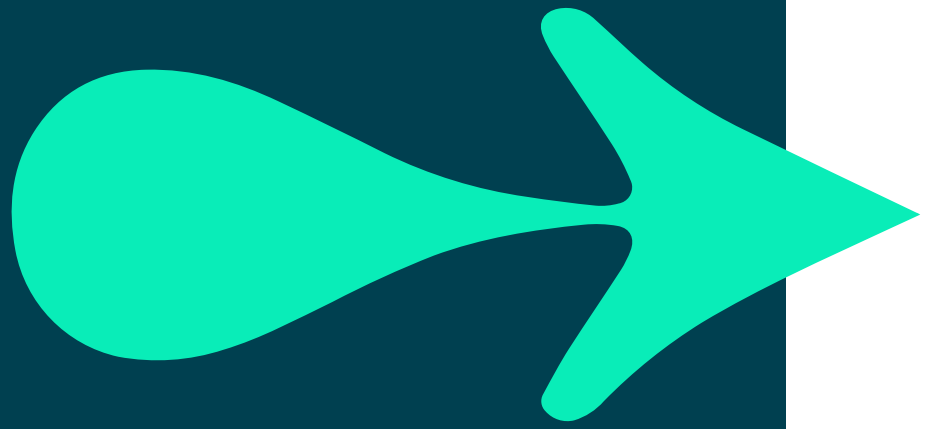
POWERING POTENTIAL

Conditionals

Module 10

FLOW CONTROL



Contents

Python conditionals

Conditional membership is by *indentation*

- Designed for readability
- *Syntax:*

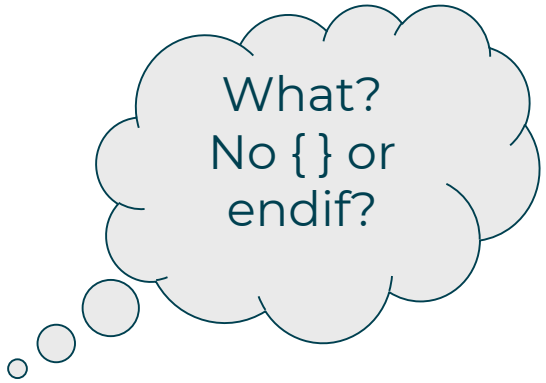
```
if condition:
    statements
elif condition:
    statements
else:
    statements
```

- Boolean operators are overloaded by type

→ **No need for different text or numeric operators**

```
if lista == listb:
    print("Same!")

if "eggs" in lista:
    print("It eggists!")
```



What is truth?

Built-in function `bool()` tests an object as a Boolean

- False : 0, `None`, empty string, tuple, list, dictionary, set
- True : everything else
- Constants True and False are defined

py3

True and False are not constants in Python 2!

Use double equal signs (==) to compare values

- Overloaded for built-in types
- Use `is` to compare identities of two objects

Sequence types and dictionaries also support `in`

- Tests membership of the container

```
lang = ['Perl', 'Python', 'PHP', 'Ruby']
if 'Python' in lang:
    print('Python is there')
```

py3

`in` was introduced at 2.6

Boolean and logical operators

Boolean operators

`<`	**value less than**	*expression* `<` *expression*
`<=`	**value less than or equal**	*expression* `<=` *expression*
`>`	**value greater than**	*expression* `>` *expression*
`>=`	**value greater than or equal**	*expression* `>=` *expression*
`==`	**value equality**	*expression* `==` *expression*
`!=`	**value inequality**	*expression* `!=` *expression*
`is`	**object identity is the same**	*object* `is` *object*

Python 2 also had <> for value inequality

Logical operators

`not`	**logical NOT**	`not` *expression*
`and`	**logical AND**	*expression* `and` *expression*
`or`	**logical OR**	*expression* `or` *expression*

Chained comparisons

- Useful for a testing a range of values

```
if 0 < number < 42 < distance:
    print("number and distance are within range")
else:
    print("number and distance are out of range")
```

- Same as:

```
if 0 < number and number < 42 and 42 < distance:
    print("number and distance are within range")
else:
    print("number and distance are out of range")
```

- Can be combined

```
if 0 < number < 42 and distance != 20:
    ...
```

Sequence and collection tests

- **An empty string, tuple, list, dictionary, set returns False**

```
mylist = [0, 1, 2, 3]
if mylist:
    print("mylist is True")
```

```
mylist is True
```

- **Sequences also support built-in `all` and `any`**
  - **`all`** returns True if all items in the sequence are true
  - **`any`** returns True if *any* of the items in the sequence are true

```
mylist = [0, 1, 2, 3]

if not all(mylist):
    print("mylist: not all are True")

if any(mylist):
    print("mylist: at least one item is True")
```

```
mylist: not all are True
mylist: at least one item is True
```

Object types

Beware of comparing objects of different types

- Comparison operators may be overloaded
- Do not expect automatic conversion

```
num = 42
txt = '3'

if txt < num:
    print('Wow!')
else:
    print('Doh!')

if int(txt) < num:
    print('Wow!')
else:
    print('Doh!')
```

py3

```
TypeError: unorderable
types: str() < int()
```

Before Python 3 this was not an error, we just got the wrong answer!

```
Wow!
```

A note on exception handling

- **An Exception is Python's way of telling you something**
  - → Unless handled, it will halt the program
- **Many Python built-in functions can raise an exception**
  - → When they wish to indicate some condition
- **Exceptions do not necessarily indicate failure**
  - → For example:
    - Search for something which does not exist
    - Unable to open a file

- **At this point in the course, we will just live with them**
- **Later, we will discuss how to handle exceptions**

While loops

Loop while a condition is true

- Python only supports entry condition loops
- There is no *do...while* loop

With all conditionals, membership is by *indentation*

`while` *condition*:
    loop body

```
line = None

while line != 'done':
    line = input('Type "done" to complete: ')
    print('<', line, '>')
```

```
myl = [23, 67, 32, 9, 77]

while myl:
    print(myl.pop() * 2)
```

```
154
18
64
134
46
```

`pop()` on a list removes and returns the last item

Loop control statements

- **Loop control statements**
  - → `continue` perform next iteration
  - → `break` exit the loop at once
  - → `pass` Empty placeholder (do nothing)
- **The else: clause**
  - → Indicates code to be executed when the while condition is false, or when the for list expires
  - → Including when the loop condition is false on entry

```
i = 1
j = 120
while i < 42:
    i = i * 2
    if i > j: break
else:
    print("Loop expired: ", i)

print("Final value: ", i)
```

The `else` clause is not executed if the loop exits using a `break`

```
Loop expired:  64
Final value:  64
```

For loops

- **Iterate through a sequence**
  - → Often a list or tuple
  - → Loop variable holds a copy of each element in turn
- **As with conditionals, membership is by *indentation***

```
for variable in object:
    loop body
```

```
import sys
for arg in sys.argv:
   print("Cmd line argument:", arg)
```

`sys.argv` is a list of the command-line arguments

```
C:\Python>for.py Monday Tuesday Wednesday
Cmd line argument: C:\Python\for.py
Cmd line argument: Monday
Cmd line argument: Tuesday
Cmd line argument: Wednesday
```

enumerate

Use in loops over any sequence

- Returns a two-item tuple which contains a count and the item at that position in the sequence

```
for idx, arg in enumerate(sys.argv):
    print('index:’, idx, 'argument:', arg)
```

… or other object type which supports iteration

- For example, `open` will open a file and return an iterator
- enumerate also takes an optional *start* parameter

```
for nr, line in enumerate(open('brian.txt'), start=1):
    print(nr, line, end="")
```

line numbers start from 1, sequences start at 0

```
1 Some things in life are bad
2 They can really make you mad
3 Other things just make you swear and curse.
```

Counting 'for' loops

- **Can use the `range()` builtin**

py3 `range` used to be called `xrange`

`range`([*start*], *stop*[, *step*])

```
for i in range(0, len(some_list)):
    if some_list[i] > 42: some_list[i] += 1
```

- But this maintains its own iterator

```
for i in range(0, len(some_list)):
    print(some_list[i])
```

- Use a system generated one instead

```
for num in some_list:
    print(num)
```

- **But an index is needed to alter the sequence...**

```
for idx, num in enumerate(some_list):
    if num > 42: some_list[idx] += 1
```

Conditional expressions

- Shorthand for conditionals
  expr1 **`if`** *boolean* **`else`** *expr2*

```
i = 42
j = 3

print("i gt j") if i > j else print ("i lt j")

print("i gt j" if i > j else "i lt j")
```

```
if i > j:
    print("i gt j")
else:
    print("i lt j")
```

These 'if' statements all do the same thing

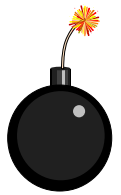
- No : and elif not allowed

```
-1 if a < b else (+1 if a > b else 0)
```

- Beware of precedence

```
a = 54
answer = a + 5 if a < 42 else 0

answer = a + (5 if a < 42 else 0)
```



Unconditional closedown

`os._exit(`*integer_expression*`)`

- Cannot be trapped
- Returns *integer_expression* to the caller (usually the shell)

`os.abort()`

- Raises a SIGABRT signal (trappable on UNIX)
- Causes a core dump on UNIX, an exit 3 on Windows

`sys.exit(`*expression*`)`

- Raises a `SystemExit` exception which can be trapped
- Returns expression to the caller (usually the shell) if it is an integer
- Prints to stderr if any other type of object
  - Returns 1 to the caller

```
sys.exit("Goodbye")
```

Unconditional flow control (2)

- ***"But I use `exit()` or `quit()`!"***
- **At start-up, the `site` module is automatically loaded**
  - Unless the -S command-line option is given
  - Several objects are created, including **`exit`** and **`quit`**
- **When *printed*, `exit` and `quit` output a message:**

```
>>> exit
Use exit() or Ctrl-Z plus Return to exit
>>> quit
Use quit() or Ctrl-Z plus Return to exit
```

- **When *called* they raise a `SystemExit` exception and close `stdin`**
  - IDLE ignores `SystemExit` but closes when `stdin` is closed
- **Only use in an interpreter session, not in production code**
  - Because of the side-effect of closing `stdin`

SUMMARY

- **Python has the usual Boolean and logical operators**
- Be careful of types
- **Basic flow control statements :**

if *condition*:

  indented statements

while *condition*:

  indented statements

for *target* in *object*:

  indented statements

- **Terminate a process using `sys.exit()`**