

COMP10001 Foundations of Computing

Conditionals and Functions

Semester 1, 2019

Tim Baldwin, Nic Geard, Farah Khan, and Marion Zalk



— VERSION: 1482, DATE: MARCH 19, 2019 —

© 2019 The University of Melbourne

Lecture Outline

① Conditionals (cont.)

② Functions

Capturing Truth: The bool Type

- We capture truth via the `bool` (short for “Boolean”) type, which takes the two values: `True` and `False`
- As with other types, we can “convert” to a `bool` via the `bool()` function:

```
>>> bool(3)
True
>>> bool(0)
False
>>> bool("banana")
True
```

Every type has a unique value for which `bool()` evaluates to `False`

Lecture Agenda

- Last lecture — Grok Worksheets 3–4
 - String manipulation
 - Conditionals
- This lecture — Grok Worksheets 3, 5
 - Conditionals (cont.)
 - Functions

In Search of the Truth ...

- For this, we require:
 - a way of describing whether the test is satisfied or not
 - a series of comparison operators
 - a series of logic operators for combining comparisons
 - a way of conditioning behaviour on the result of a given test

Evaluating Truth: Comparison

- We evaluate truth via the following Boolean comparison operators:
 - `==` equality; NOT the same as `=`
 - `>`, `>=` greater than (or equal to)
 - `<`, `<=` less than (or equal to)
 - `!=` not equal to
 - `in` is an element of

```
>>> 2 == 3
False
>>> 'a' <= 'apple'
True
>>> 'bomp' in 'bomp, bomp, bomp'
True
```

Combining Truth

- We combine comparison operators with the following logic operators:

- `and`, `or`, `not`:

<code>and</code>	True	False
True	True	False
False	False	False

<code>or</code>	True	False
True	True	True
False	True	False

<code>not</code>	True	False
	False	True

- NB: precedence: `not` > `and` > `or`

Combining Truth: Examples

- The way logic operators are interpreted in Python is by evaluating the truth value of each operand, and combining them, e.g.:

```
>>> tall and ears == "rabbit" and 3
```

is equivalent to:

```
>>> bool(tall) and bool(ears == "rabbit") and \
... bool(3)
```

Conditioning and Code Blocks

- We can condition the execution of a “block” of code with `if` statements

a “block of code” is a contiguous series of lines of code which are “indented” at (at least) a certain level

```
if balance >= 0:
    balance = balance - withdraw
    print("Withdrawn")
    if balance < 0:
        print("Time to ring mum!")
```

The block only executes if the condition in the `if` statement evaluates to `True`

Combining Truth: Examples

```
>>> age = 20
>>> age >= 18
True
>>> tall = True; ears = "rabbit"; back = "grey"
>>> whiskers = True; stomach = "cream"
>>> has_umbrella = True
>>> tall and ears == "rabbit" and back == "grey" and \
... whiskers and stomach == "cream" and has_umbrella
True
>>> not False or True
True
>>> not (False or True)
False
>>> year = 2015
>>> 2001 < year < 2100
True
```

Things that aren't as They Seem

- One comparison operator that you may run into, but **should avoid** (for now) is `is`; intuitively it may feel like it is another way of testing that two objects are comparable in value and type, but what it really tests for is whether two objects are *identical*
- Another common gotcha is complex expressions such as:

```
>>> name = 'kim'
>>> bool(name == 'sandy' or 'alex')
True
```

Why? correctly:

```
>>> name = 'kim'
>>> bool(name == 'sandy' or name == 'alex')
False
```

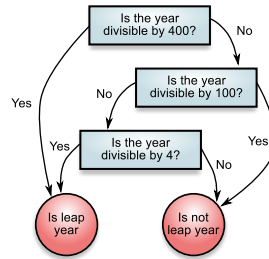
Class Exercise

What is the output of the following code:

```
a = 1
b = 5
if b:
    b = a + 1
else:
    b = b + 1
print(a, b)
```

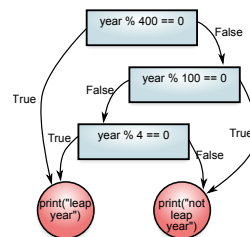
Conditional Recap

- Problem: evaluate whether a given year is a leap year (True) or not (False)
- Flowchart:



Conditional Recap

- Problem: evaluate whether a given year is a leap year (True) or not (False)
- Pythonic flowchart:



Lecture Outline

① Conditionals (cont.)

② Functions

Cascading Conditions

- It is possible to test various **mutually-exclusive** conditions by adding extra conditions with `elif`, and possibly a catch-all final state with `else`

```

if year % 400 == 0:
    print("leap year")
elif year % 100 == 0:
    print("not leap year")
elif year % 4 == 0:
    print("leap year")
else:
    print("not leap year")
  
```

Class Exercise

- Simplify the preceding code into one `if` statement and one `else` statement (and no `elif` statements)

Functions: Introduction

- What's a function?
 - (much like in Maths) functions take a set of input values, perform some calculation based on them, and return a value
 - you have already seen and used a smattering of functions by this stage, e.g.: `str()`, `len()`, ...
- Wouldn't it be nice to be able to recycle chunks of our own code?

Functions: The Details

- In order to define a function, we need:
 - A function name (following same conventions as other variable names)
 - (optionally) a list of input parameters
 - some code to actually execute (the “body” of the function)
 - (optionally) a UNIQUE output object (via `return`)

- Basic form:

```
def NAME(INPUTLIST):
    statement_block
```

NB: the characters here indicate space characters

The Power of return

- In order to use the output of a function (e.g. to assign it to a variable), we need to `return` a value:
- Convert from Celsius to Fahrenheit:

```
def C2F(n):
    return 9*n/5 + 32
print(C2F(21))
```

- Count the digits in a number:

```
def print_digits(num):
    return len(str(abs(num)))
print(count_digits(-123))
```

Class Exercise

What is printed here?

```
def bloodify(word):
    return word[:3] + '-bloody-' + word[3:]

print(bloodify('fantastic'))
print(bloodify('marion'))
```

Warm-up Functions

- Convert from Celsius to Fahrenheit:

```
def print_C2F(n):
    print(9*n/5 + 32)
```

- Count the digits in a number:

```
def print_digits(num):
    print(len(str(abs(num))))
```

The Power of return

- `return` is also a way of (unconditionally and irrevocably) terminating a function:

```
def safe_divide(x,y):
    if y:
        return x/y

    print("ERROR: denom must be non-zero")
```

Lecture Summary

- What logic operators are commonly used in Python? What is the operator precedence?
- What are `if` statements and code blocks?
- How can you cascade conditions in Python?
- Comments: what and how?
- What is a function, and what is its basic form?
- What does `return` do?