COMP10001 Foundations of Computing
Conditionals and Functions

Semester 2, 2021

Chris Leckie, Marion Zalk and Farah Khan

— VERSION: 1484, DATE: MARCH 21, 2019 —

Lecture Agenda

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

Lecture Outline

1. Conditionals (cont.)

2. 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

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`

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`:

`and`	`True`	`False`
`True`	`True`	`False`
`False`	`False`	`False`

`or`	`True`	`False`
`True`	`True`	`True`
`False`	`True`	`False`

`not`	`True`	`False`
	`False`	`True`

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

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
```

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)
```

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
```

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    print("Withdrawn")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

The block only executes if the condition in the `if` statement evaluates to `True`. There can also be an optional `else` statement with a block of code to be executed if the condition is `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)
```

Class Exercise

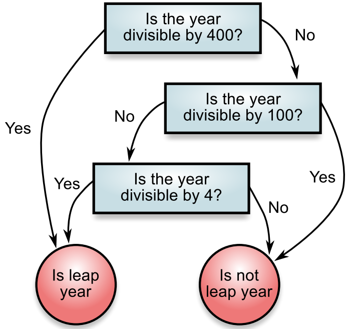
What is the output of the following code:

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

We can try visualising the execution of this code using a handy website called `http://www.pythontutor.com`

Conditional Recap

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


Is the year divisible by 400?
No
Is the year divisible by 100?
Yes
No
Is the year divisible by 4?
Yes
Yes
No
Is leap year
Is not leap year

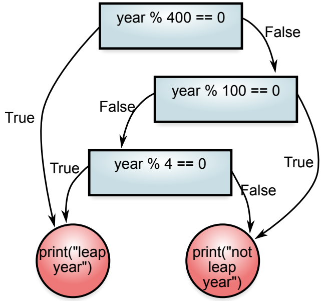
Cascading Conditions

- 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")
```

Conditional Recap

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



Class Exercise

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

Lecture Outline

1. Conditionals (cont.)
2. Functions

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

Warm-up Functions

- Convert from Celsius to Fahrenheit:

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

- Count the digits in a number:

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

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

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")
```

Class Exercise

What is printed here?

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

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

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?
- What is a function, and what is its basic form?
- What does `return` do?