8 Conditionals

Key terms: `if elif else`

Reading: Severance 3

The procedural or imperative model of programming is the one most programmers already know, with looping and branching. This approach runs the risk of creating "spaghetti code" — programs where execution jumps around in complicated ways. It was thought that "structured" procedural code would resolve the problems associated with "goto" statements, but stateful nested loops can also be very confusing.

The hierarchical structure of Python code is delimited by full colons and indentation, instead of curly brackets. Code blocks are set off by a trailing full colon followed by additional indentation, and end when their indentation ends.

For conditionals, mutually exclusive branches live within the blocks of one `if`, any number of `elif`s, and possibly one `else`. Program flow to each branch is controlled by a condition, which is a `bool`, either accessed in memory or calculated on the spot. Consider the object produced by the expression `2>1`; it's `True`.

```
if x == 1:
 print("Perfect!")
elif x > 1:
 print("High.")
else:
 print("Low.")
```

The order of a series of conditions matters because each one excludes the rest. The branches are mutually exclusive, and therefore any order should work as well as another. We usually prefer clarity as a criterion for order over performance. However, branching conditions which are not truly mutually exclusive make the order positively matter, which can be difficult to notice. Take care that your conditions be mutually exclusive.

`if`/`elif`/`else` branches written in procedural, block style require many lines of code. This single line with a nested ternary conditional has the same functionality as the above:

```
print("Perfect!" if x == 1 else "High." if x > 1 else "Low.")
```

Note that objects which are not exactly `bool`s can nevertheless have Boolean meanings for the purpose of branching. With regard to the length of a string, instead of `if len(name)>0`, we can write `if len(name)`, or indeed even `if name`, to do the same job. A special operator precedence for "chained" comparisons allows the use of compounds like `2005<year<2015`.

```
# Assess a person's age and voting rights:

if age <= 1:
 cohort = 'baby'
elif 2 <= age <= 12:
 cohort = 'child'
elif 13 <= age <= 19:
 cohort = 'teenager'
elif 20 <= age <= 69:
 cohort = 'adult'
else:
 cohort = 'senior'

voter = True if age > 18 else False

print(age,cohort,voter)
```

Next, we will address the implications of Pythonic style to loops and program flow.