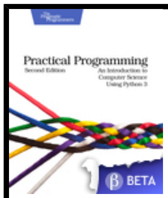

- [Chapter 5.2 Choosing Which Statements to Execute]
- [Chapter 3.8 Omitting a Return Statement: `None`]
- *Optional reading*

The `if` statement

If statements can be used to control which instructions are executed. Here is the general form:

```
if expression1:
    body1
[elif expression2:      0 or more clauses
    body2]
[else:                  0 or 1 clause
    bodyN]
```

`elif` stands for "else if", so this forms a chain of conditions.

To execute an `if` statement, evaluate each expression in order from top to bottom. If an expression produces `True`, execute the body of that clause and then skip the rest open 3he `if` statement. If there is an `else`, and none of the expressions produce `True`, then execute the body of the `else`.

For example, given this function:

```
def report_status(scheduled_time, estimated_time):
    """ (float, float) -> str """
    if scheduled_time == estimated_time:
        return 'on time'
    elif scheduled_time > estimated_time:
        return 'early'
    else:
        return 'delayed'
```

In the shell:

```
>>> report_status(14.3, 14.3)
'on time'
>>> report_status(12.5, 11.5)
'early'
>>> report_status(9.0, 9.5)
'delayed'
```

A note on `None`

When execution of a function body ends without having executed a `return` statement, the function returns value `None`. The type of `None` is `NoneType`.

For example, consider this function:

```
def report_status(scheduled_time, estimated_time):
    """ (float, float) -> str
    Return the flight status (on time, early, delayed) for a flight that was
    scheduled to arrive at scheduled_timed, but is now estimated to arrive
    at estimated_time.
    Pre-condition: 0.0 <= scheduled_time < 24.0 and 0.0 <= estimated_time < 24.0
    >>> report_status(14.3, 14.3)
    'on_time'
    >>> report_status(12.5, 11.5)
    'early'
    >>> report_status(9.0, 9.5)
    'delayed'
    """
    if scheduled_time == estimated_time:
        return 'on time'
```

In the shell:

```
    >>> report_status(14,3, 14.3)
    'on time'
    >>> report_status(12.5, 11.5)
    >>> print(report_status(12.5, 11.5))
    None
```

Because the type of `None` is `NoneType`, not `str`, this breaks the Type Contract. To fix this, we would need to complete the rest of the function.

Jennifer Campbell ? Paul Gries
University of Toronto