Data Types

- Number
- String
- Image
- Boolean

Named after George Boole, 19th century mathematician and logician.
Boolean Values

• There are:
  – Many numbers
  – Many strings
  – Many images

• But only two Boolean Values:
  true
  false
Operations on Boolean Values

Negation: `not`

```
>>> not(true)
false
```

```
>>> not(false)
true
```

```
Operations on Boolean Values
Conjunction: `and`

```python
>>> true and true
true
true
>>> true and false
false
false
>>> false and true
false
false
>>> false and false
false
false
```
Operations on Boolean Values
Disjunction: \texttt{or}

\begin{verbatim}
>>> true or true
true
>>> true or false
true
>>> false or true
true
>>> false or false
false
\end{verbatim}
Operations that Create Boolean Values
Equal: `==`

```python
>>> "foo" == "foo"
true

>>> "foo" == "bar"
false

>>> (2 + 4) == (4 + 2)
true

>>> (2 / 4) == (4 / 2)
false
```
Operations that Create Boolean Values

Less Than: <          Greater Than: >
Less or Equal: <=    Greater or Equal: >=

```python
>>> 13 < 137
true
>>> 137 < 13
false
>>> 21 <= 42
true
>>> 21 <= 21
true

>>> "zebra" > "aardvark"
true
>>> "aardvark" < "zebra"
true
>>> "DAD" < "dad"
true
>>> "dad" < "DAD"
false
>>> "dad" < "dada"
true
```
Operations that Create Boolean Values

```
>>> num-equal(2, 1 + 1)
true

>>> string-equal("foo", "bar")
false

>>> string-contains("foo", "foobar")
false

>>> string-contains("foobar", "foobar")
true
```

Why should one use `num-equal` or `string-equal` rather than `==`?
AWD Surcharge

Why define constants? (Change prices in one place in program.)
Why put these definitions at the top? (Easy reference.)
Conditional Expression:
if ... else ... end

What part of the code handles SUVs?
One must read the code above the else clause.
Conditional Expression:
if ... else if ... end

In this version one can see how SUVs are handled by looking only at the else-if clause.
AWD Surcharge

```plaintext
sedan-awd-surcharge = 1000
suv-awd-surcharge = 2000

fun awd-surcharge(vehicle :: String) -> Number:
  if (vehicle == "sedan"):
    sedan-awd-surcharge
  else if (vehicle == "suv"):
    suv-awd-surcharge
  end

where:
  awd-surcharge("sedan") is sedan-awd-surcharge
  awd-surcharge("suv") is suv-awd-surcharge
end
```
Computing Marginal Tax Rates

```language
fun marginal-tax-rate-rate(income :: Number) -> Number:
  doc: "Marginal tax rate based on income."
  if income <= 20000: 0.0
  else if income <= 50000: 0.1
  else if income <= 100000: 0.3
  else: 1.0
end

where:
  marginal-tax-rate-rate(15000) is 0.0
  marginal-tax-rate-rate(20000) is 0.0
  marginal-tax-rate-rate(35000) is 0.1
  marginal-tax-rate-rate(50000) is 0.1
  marginal-tax-rate-rate(70000) is 0.3
  marginal-tax-rate-rate(100000) is 0.3
  marginal-tax-rate-rate(125000) is 1.0
end
```

Notice that each else clause depends on clauses above it. This is efficient but hard to read and understand.

Tests include boundary cases and cases in between boundaries.
Notice that each else clause describes a time range in terms of upper and lower bounds – not depending on previous clauses. This is less efficient, but easier to read and understand.
Rock Paper Scissors

- Rock smashes scissors
- Scissors cuts paper
- Paper wraps rock.
fun rsp1(first :: String, second :: String) -> Number:
    if (first == "rock") and (second == "scissors"): 1
    else if (first == "scissors") and (second == "paper"): 1
    else if (first == "paper") and (second == "rock"): 1
    else: 2
end

where:
    rsp1("rock","scissors") is 1
    rsp1("rock","paper") is 2
end
fun rsp2(first :: String, second :: String) -> Number:
    if (first == second): 0
    else if (first == "rock") and (second == "scissors"): 1
    else if (first == "scissors") and (second == "paper"): 1
    else if (first == "paper") and (second == "rock"): 1
    else if (second == "rock") and (first == "scissors"): 2
    else if (second == "scissors") and (first == "paper"): 2
    else if (second == "paper") and (first == "rock"): 2
end

where:
    rsp1("rock","scissors") is 1
    rsp1("rock","paper") is 2
end