This course will be in R, not Python

**Slide 20:**

Notice the default panes:

\* Console (entire left)

\* Environment/History (tabbed in upper right)

\* Files/Plots/Packages/Help (tabbed in lower right)

FYI: you can change the default location of the panes – go to settings

Go into the Console on the left with the `>` (that's the \*command prompt\*), and let's get started!

**Slide 25:**

## Odds and evens with `%%`

You can tell if an integer is even or odd by using mod `2`. If you get `0`, it's even; if you `1`, it's odd:

```{r}

10 %% 2 # Even

11 %% 2 # Odd

12 %% 2 # Even

13 %% 2 # Odd

```

**Slide 26:**

You can also apply equality operations to strings.

```{r}

"penguin" == "penguin"

```

However, R is very particular about what counts as equality. For two pieces of text to be equal, they must be \_precisely\_ the same:

```{r, results='hold'}

"penguin" == "PENGUIN" # Returns FALSE

"penguin" == "p e n g u i n" # Returns FALSE

"penguin" == "penguin " # Returns FALSE

```

**Slide 27:**

## Logical operators

R assesses logical statements using the following logical operators:

- And: `&`

- Or: `|`

- Not: `!`

\*\*And\*\*:

A logical expression `x & y` is `TRUE` only if \*both\* `x` and `y` are `TRUE`.

```{r}

(2 == 2) & (2 == 3)

```

```{r}

(2 == 2) & (3 == 3)

```

\*\*Or\*\*:

A logical expression `x | y` is `TRUE` if \*either\* `x` or `y` are `TRUE`.

```{r}

(2 == 2) | (2 == 3)

```

\*\*Not\*\*:

The `!` operator behaves like the word \*"not"\* in everyday language. If a statement is "not true", then it must be "false". For example:

```{r}

!TRUE

```

In practice, it is often helpful to include parentheses to clarify the statement being made. Consider the following example:

```{r}

!3 == 5

```

This returns `TRUE`, but it's a bit confusing. Reading from left to right, you start by saying "not 3"...what does that mean?

What is really going on here is R first evaluates whether 3 is equal to 5 (`3 == 5`), and then returns the "not" (`!`) of that. A better version of the same thing would be:

```{r}

!(3 == 5)

```

**Slide 28:**

```{r}

1 + 2 \* 4

```

```{r}

(1 + 2) \* 4

```

**Slide 36:**

round(3.1415)

round(3.1415, digits=2)

**Slide 39:**

The operator `<-` is a pain to type, so RStudio has a keyboard shortcut: Alt + -

\*\*R is case sensitive\*\*. To understand what this means, try this:

**Slide 40:**

## Numeric types

By default, R assumes all numbers are \*\*floats\*\*, even if it \_looks\_ like an integer:

```{r}

class(3.14)

class(3)

typeof(3.14)

typeof(3)

```

The fact that `3` is not an integer can be confirmed with the `is.integer()`function:

```{r}

is.integer(3)

```

To make a number an integer, you have to either add an `L` to the end, or use the `as.integer()` function to coerce it into an integer type:

```{r}

typeof(3L)

typeof(as.integer(3))

```

Note that you can have a decimal of `0` but the data types can be different:

```{r}

typeof(7)

typeof(7.0)

typeof(7L)

```

## Character types

A character object is used to represent string values in R. Anything put between single quotes (`''`) or double quotes (`""`) will be stored as a character. For example:

```{r}

s <- '3.14'

class(s)

```

Notice that even though the value assigned to `s` is a number, it is stored as a character because the number is surrounded by quotes. If you mistakenly thought it was stored as a number, R will gladly return an error when you try to do a numerical operation with it:

```{r error=TRUE}

s + 7

```

We can also convert objects into character values with the `as.character()` function:

```{r}

a\_number <- 3.14

class(a\_number)

a\_string <- as.character(a\_number)

class(a\_string)

```

We will go into much more detail about character data later on in [Week 6](L6.1-strings.html)

## Logical types

A logical value is usually created when comparing variables. For example, let's compare whether one value is greater than another:

```{r}

x <- 1

y <- 2

z <- x > y # Is x greater than y?

z

```

The object `z` now stores the value (`FALSE`) that resulted from comparing `x` and `y`.

```{r}

x == y

```

Here I am asking, "is `x` equal to `y`"? In this case, the answer is `FALSE`.

We will go into more detail about comparing values in R later on in [Week 4](L4.1-conditionals.html)

## Special values

In addition to the four main data types mentioned, there are a few additional "special" types: `Inf`, `NaN`, `NA` and `NULL`.

\*\*Infinity\*\*: `Inf` corresponds to a value that is infinitely large. You can also have `-Inf`. The easiest way to get `Inf` is to divide a positive number by 0:

```{r}

1/0

```

\*\*Not a Number\*\*: `NaN` is short for "not a number", and it's basically a reserved keyword that means "there isn't a mathematically defined number for this." For example:

```{r}

0/0

```

\*\*Not available\*\*: `NA` indicates that the value that is "supposed" to be stored here is missing. We'll see these much more when we start getting into data structures like vector, matrices, and data frames.

\*\*No value\*\*: `NULL` asserts that the variable genuinely has no value whatsoever, or does not even exist.

**Slide 42:**

## Coercing value types

In general, you can force a value into a different type using `as.something()`, replacing "`something`" with a data type:

```{r}

as.numeric(3.1415)

as.integer(3.1415)

as.character(3.1415)

as.logical(3.1415)

```

Note that `as.logical()` returns `TRUE` for any numeric value other than `0`, for which it returns `FALSE`.

If you try to coerce a string into a number, R will return `NA`:

```{r}

as.numeric('foo')

```

**Slide 44:**

```{r eval=FALSE}

install.packages("wikifacts") # Remember - you only have to do this once!

```

```{r}

library(wikifacts) # Load the library

```

```{r}

wiki\_randomfact()

wiki\_didyouknow()

```