**UNIT 1 READING NOTES**

**Chapter 2: Names and Values**

* What are the rules for object names, and what happens if you try to use a not-allowed name with <- or with make.names()?
  + A **syntactic** name must consist of letters10, digits, . and \_ but can’t begin with \_ or a digit. Additionally, you can’t use any of the **reserved words** like TRUE, NULL, if, and function
  + If you try to use a not-allowed name, R will say unexpected input or unexpected output in “ “
* What does *copy-on-modify* mean and why is it important for designing fast/efficient code?
  + Modifying y clearly didn’t modify x. So what happened to the shared binding? While the value associated with y changed, the original object did not.
    - Modifying a copy of a values in R creates a new value and it makes sense because its faster to create a new value rather than overwrite another since it’s immutable.
* Why might the size of a *list*seem surprisingly small, even if it contains large objects?
  + This list is more complex because instead of storing the values itself, it stores references to them:
    - As seen by this excerpt, the list appears to be small because it doesn’t contain the actual large objects, but just references to them.
* When data frames column are modified only the column gets modified and the data frame references that modified column. But if multiple columns are modified than the entire data frame is modified.
* Modify-in-place:
  + R creates a copy, but here are two exceptions:
* Objects with a single binding get a special performance optimisation.
* Environments, a special type of object, are always modified in place.
* Garbage Collector
* Tracing

**Chapter 3: Vectors**

* What is the relationship between vectors, lists, matrices, and data frames?
* What are *attributes*?  How can you modify them?
  + “named list of arbitrary metadata”
  + Modify atomic vectors by adding attributes to create matrices, arrays, factors, or date-times.
* What is special about the attribute called "class"?
  + Class is special because it turns objects into s3 vectors, factors, dates, and date-times.
* Two attributes that are routinely preserved are:
  + **names**, a character vector giving each element a name.
  + **dim**, short for dimensions, an integer vector, used to turn vectors into matrices or arrays.
* What is dangerous about stringsAsFactors?  (Hint: What is the *type* for factors?)
  + It’s dangerous because R does not know the order or set of all possible levels.

In this section, we’ll discuss four important S3 vectors used in base R:

* Categorical data, where values come from a fixed set of levels recorded in **factor** vectors.
* Dates (with day resolution), which are recorded in **Date** vectors.
* Date-times (with second or sub-second resolution), which are stored in **POSIXct** vectors.
* Durations, which are stored in **difftime** vectors.

A diagram of different types of numbers

Description automatically generated

* List are a complexity step up from atomic vectors.

There are three reasons why row names are undesirable:

* Metadata is data, so storing it in a different way to the rest of the data is fundamentally a bad idea. It also means that you need to learn a new set of tools to work with row names; you can’t use what you already know about manipulating columns.
* Row names are a poor abstraction for labelling rows because they only work when a row can be identified by a single string. This fails in many cases, for example when you want to identify a row by a non-character vector (e.g. a time point), or with multiple vectors (e.g. position, encoded by latitude and longitude).
* Row names must be unique, so any duplication of rows (e.g. from bootstrapping) will create new row names. If you want to match rows from before and after the transformation, you’ll need to perform complicated string surgery.

**Chapter 4: Subsetting**

* What are the six ways of subsetting vectors?
  + **Positive integers** return elements at the specified positions:
  + **Negative integers** exclude elements at the specified positions:
  + **Logical vectors** select elements where the corresponding logical value is TRUE. This is probably the most useful type of subsetting because you can write an expression that uses a logical vector:
  + **Nothing** returns the original vector. This is not useful for 1D vectors, but, as you’ll see shortly, is very useful for matrices, data frames, and arrays. It can also be useful in conjunction with assignment.
  + **Zero** returns a zero-length vector. This is not something you usually do on purpose, but it can be helpful for generating test data.
  + If the vector is named, you can also use **character vectors** to return elements with matching names.
* When should you use drop = FALSE?
  + To preserve original dimensionality
* What is *recycling* and in what circumstances does it occur?
  + Recycling means reusing values in the shorter length vector when doing an operation.
  + If x and y are different lengths. The shorter of the two are recycled to the length of the longer one.
* Subsetting a single element:
  + [[[](https://rdrr.io/r/base/Extract.html) is most important when working with lists because subsetting a list with [[](https://rdrr.io/r/base/Extract.html) always returns a smaller list.
  + [$](https://rdrr.io/r/base/Extract.html) is a shorthand operator: x$y is roughly equivalent to x[["y"]]. It’s often used to access variables in a data frame

A screenshot of a computer error

Description automatically generated

* Subsetting additional:
  + There are two additional subsetting operators, which are needed for S4 objects: [@](https://rdrr.io/r/base/slotOp.html) (equivalent to [$](https://rdrr.io/r/base/Extract.html)), and slot() (equivalent to [[[](https://rdrr.io/r/base/Extract.html)). [@](https://rdrr.io/r/base/slotOp.html) is more restrictive than [$](https://rdrr.io/r/base/Extract.html) in that it will return an error if the slot does not exist. These are described in more detail in Chapter [15](https://adv-r.hadley.nz/s4.html#s4).
* What are the functions pluck() and chuck() useful for?
  + When the element is missing, pluck() always returns NULL (or the value of the .default argument) and chuck() always throws an error.
* What is Boolean masking (aka Boolean algebra) and when is it useful?  When might it be overkill?
  + Boolean masking is logical subsetting
    - It’s useful when easily combine conditions from multiple columns
  + Might be overkill when we should use set operations instead:
    - You want to find the first (or last) TRUE.
    - You have very few TRUEs and very many FALSEs; a set representation may be faster and require less storage.