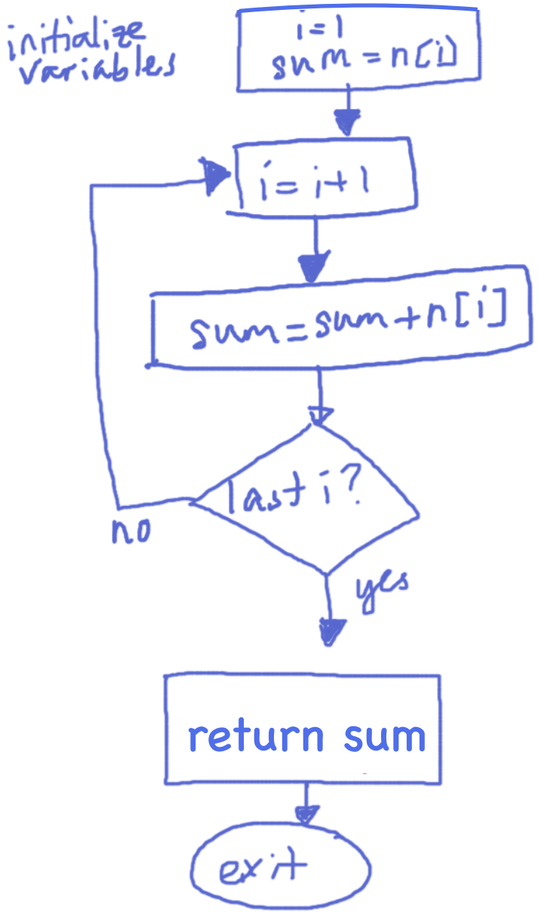
Program Flow

Abstract your programming into distinct actions or **steps**

What has to happen **within** each **step**?

How are the **steps connected**?

Simpler Example

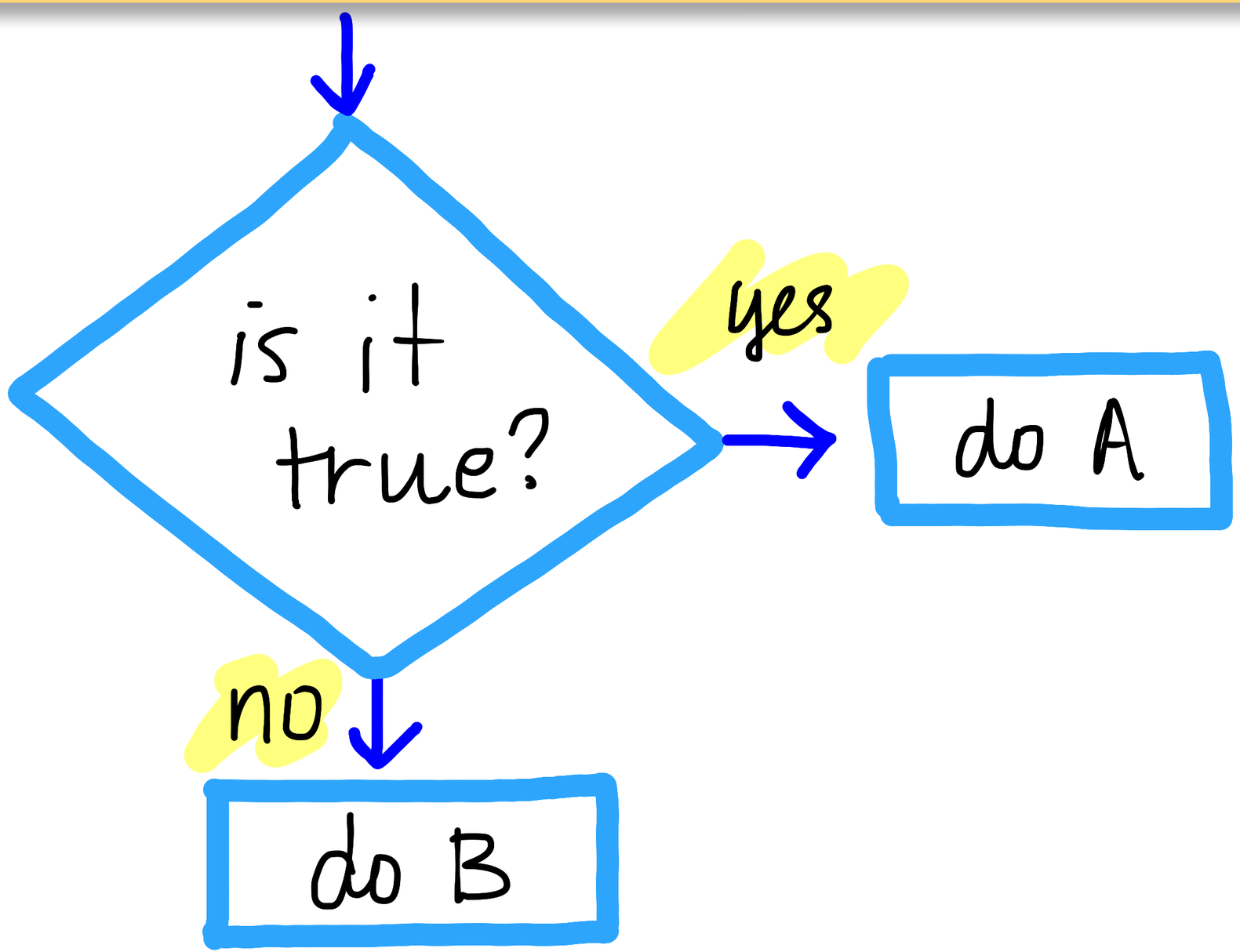


Flowchart for computing sum using a loop

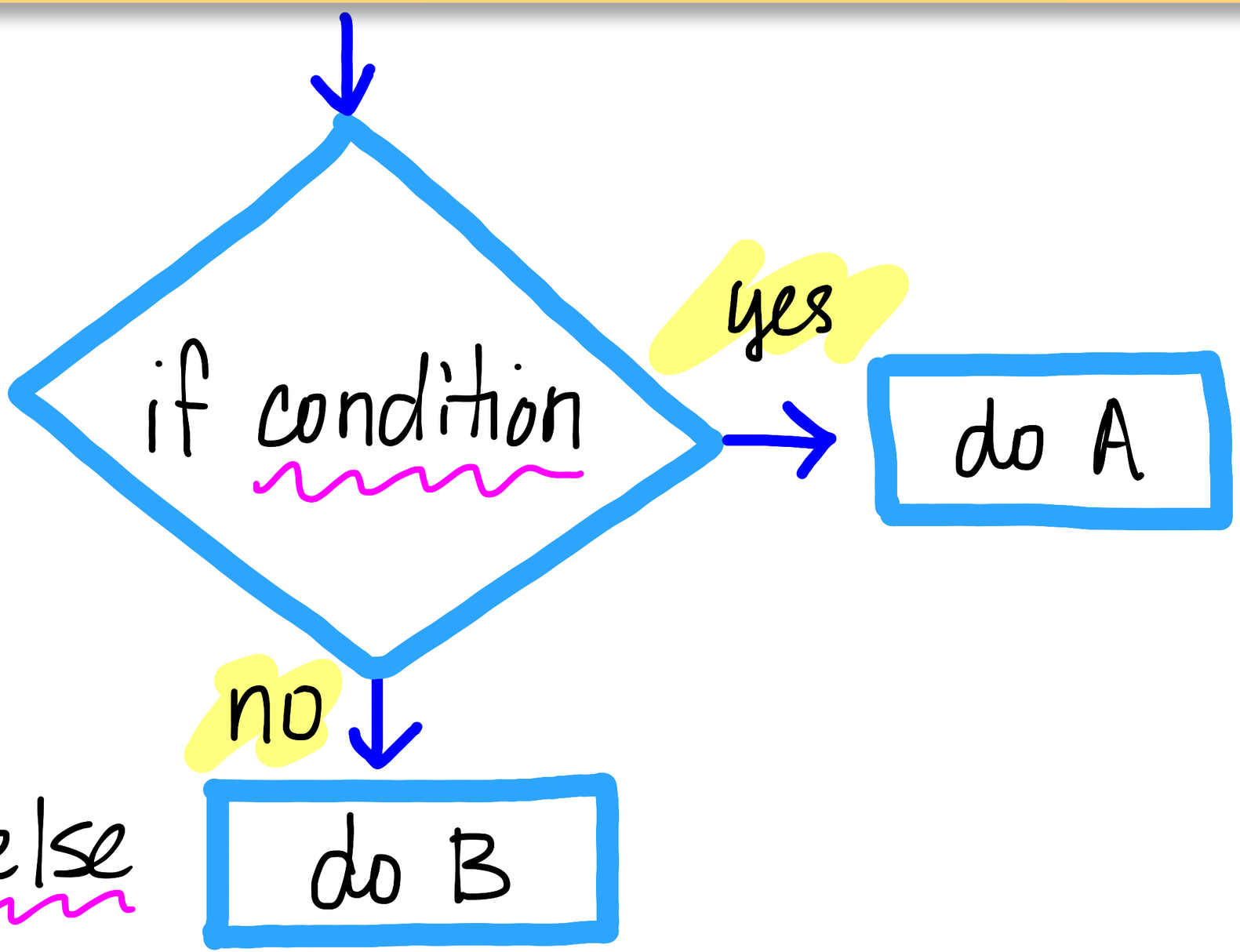
Conditional Statements

`if else`	statement
`while`	loop
`repeat`	loop

Conditional Statements `if else` statement



Conditional Statements `if else` statement



Conditional Statements `if else` statement

if condition statement else statement

```
if   x>y   print(x)
```

else is optional - nothing happens if condition false

```
if   x>y   print(x) else print ("x is too small")
```

if true

else is what happens if condition is false

Conditional Statements

while loop

```
while ( condition ) expression
```

while condition is TRUE allows execution

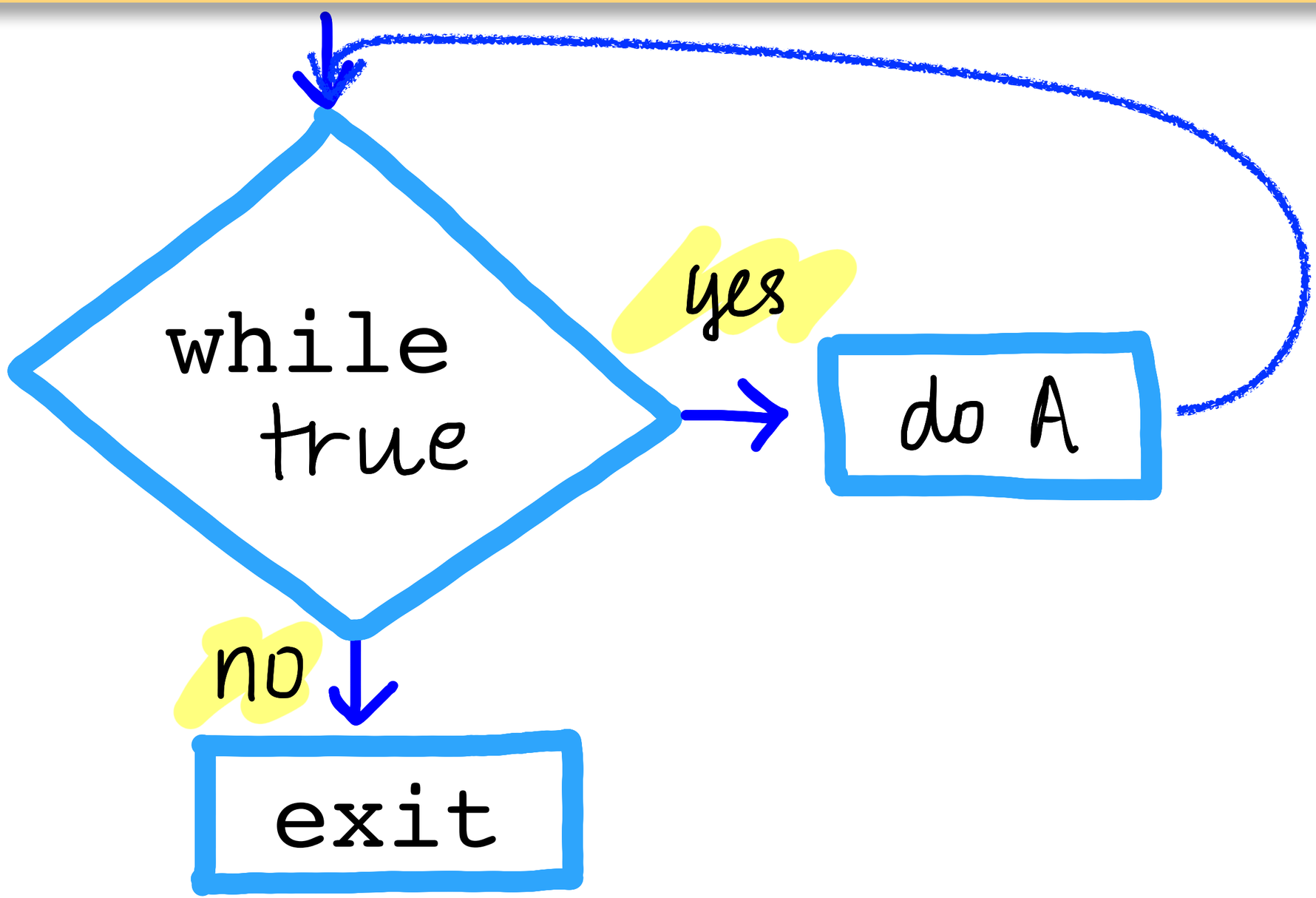
```
while ( tolerance > .001 ) {
  do some more calculations
}
```

tests before executing the expression

so make sure the condition can change (and be FALSE), or you will have an **infinite loop!**

Conditional Statements

while loop



Conditional Statements — repeat loop

```
repeat ( condition ) expression


repeat {

  expressions
  if (condition) { break }

}
```

break is required to stop execution.

Need conditional as well.

Be Careful!!

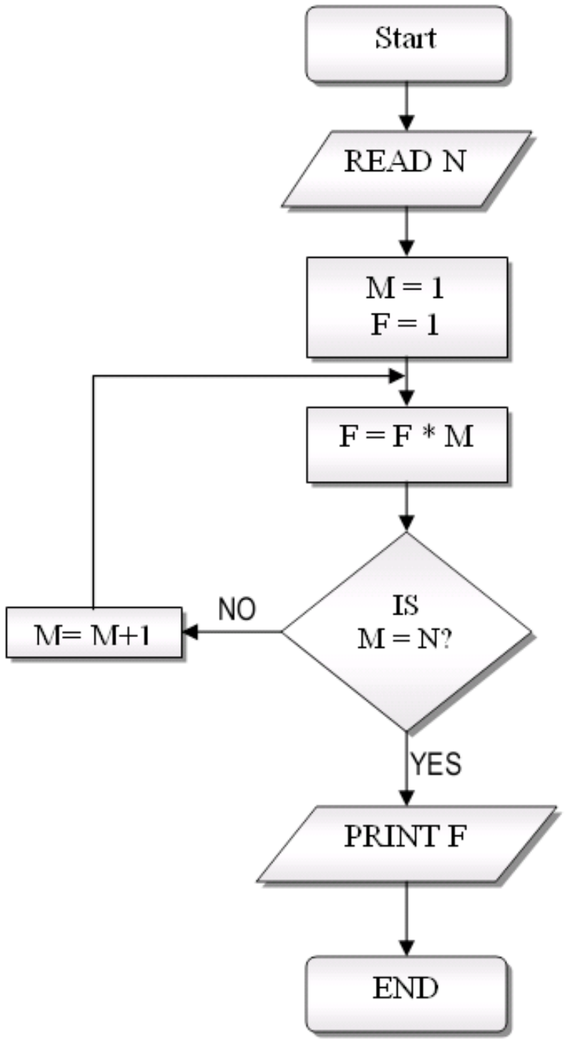
In general, try to avoid while and repeat loops if there is another way.

They can go on forever if you're not careful.

Program Flow

computing a factorial 4*3*2*1

flowchart



pseudo code

target number

initialize objects, start from 1

multiply current with next

are we at target?

yes, fantastic. stop.
no: add 1 and go back to

Vectorized Calculations apply functions

```
x <- matrix( 1:4, nrow=2)
x + 1
```

many simple functions in R are already "vectorized"

Apply allows you to "apply" functions to an entire list, vector, row, or column

```
apply (X, MARGIN, FUN )
```

```
dat <- iris[1:4,]
apply( dat, 2, mean )
```

"2" is the row index, and function is mean, so taking mean over rows

Vectorized Calculations apply functions

Flavors

tapply operates on a "ragged array" (i.e., groups of different sizes, for instance one object indexed by a list of factors)

note: index must be coerced to list

```
tapply (array or vector, index , FUN)
```

```
tapply( iris$Petal.width, list(iris$Species), mean )
                                    # calculate means by species
```

Vectorized Calculations `apply` functions

Flavors

these all operate on components of a list or vector or array

```
apply      #operates over margin (1=rows, 2=columns)
lapply     # returns a list
sapply     # returns "friendly" output, vector or matrix if possible
mapply     # works on multiple  arguments — HEADACHES!
aggregate  # computes summary statistics over subsets of the data
```