Introduction to

- Language
- Computer program
- For data analysis and graphics

- Free and Open-Source
- Builds upon existing software (C, Java, …)
- Contributions by many (researchers)
- Cutting-edge and pervasive
- Expanding & improving

*P*ython is for *P*rogrammers

R is for *R*esearchers

(Most) computer languages (including R) have

- Defined basic data types (including numbers)
- Arithmetic (add, subtract, multiply, divide)
- Conditional loop (for / while)
- Branch (if / else)
- Reuse code (functions)

- File input/output
- Library of functions
- Allow for creation of complex data types

Primitive data types (are vectors)

```
> x <- 32.0
> x
[1] 32
> class(x)
[1] "numeric"

> x <- as.integer(x)
> x
[1] 32
> class(x)
[1] "integer"
```

```
> x <- "32"
> class(x)
[1] "character "


> x <- as.factor(x)
> x
[1] 32
Levels: 32
> class(x)
[1] "factor"
```

Calculator

```
>
> 3 * 3
[1] 9
>
> sqrt(9)
[1] 3
>
```

```
>
>   a <- 4
>   b <- 6
>   a * b
[1] 24
>
```

Vectors

```
> a <- c(1, 2, 3, 4, 5)
> a
[1] 1 2 3 4 5

> b <- c(2, 2, 2, 2, 2)

> a * b
[1]  2  4  6  8 10

> A
Error: object 'A' not found
>
```

```
> a <- 1:5
> b <- rep(2, times=5)


> a * b
[1]  2  4  6  8 10


> sum(a*b)
[1] 30
>
```

Matrix

```
> a <- 1:5
> b <- rep(2, times=5)
> m <- cbind(a, b, x=a*b)
> m
     a b     x
[1,] 1 2     2
[2,] 2 2     4
[3,] 3 2     6
[4,] 4 2     8
[5,] 5 2    10

> class(m)
[1] "matrix"
```

Matrix Indexing

```
> m <- matrix(1:12, ncol=4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
>
> nrow(m)
> ncol(m)
> colnames(m)
```

```
> m[2,2]
[1] 5

> m[2,]
[1]  2  5  8 11

> m[,2]
[1] 4 5 6


> m[2, c(1, 3:4)]
[1]  2 8 11
```

data.frame

```
> x <- c(21,13,5)
> y <- c('a', 'b', 'c')
> d <- data.frame(x, y)


> d
   x y
1 21 a
2 13 b
3  5 c
```

list

```
> x <- c(21,13,5)
> y <- c('a', 'b', 'c')
> d <- list(x, y)

> d
[[1]]
[1] 21 13  5

[[2]]
[1] "a" "b" "c"
```

To loop or not to loop?

```
# we can do:
x <- 1:10
n <- length(x)
for (i in 1:n) {
     x[i] <- x[i] * 2
}
x
[1]  2  4  6  8 10
     12 14 16 18 20

# better:
x <- 1:10
x * 2
```

Vectory math

```
x <- 1:10
y <- 0
for (v in x) {
     y <- y + v
}
y
[1] 55
```

```
x <- 1:10
sum(x)
```

Vectory math

```
x <- 1:10
y <- 0
for (v in x) {
     y <- y + v
}
y
[1] 55
```

```
x <- 1:10
sum(x)
```

If, else

```
x <- 1:10
for (i in 1:length(x)) {
   if (x[i] < 4) {
         x[i] <- 0
   } else if (x[i] < 8) {
         x[i] <- 1
   } else {
         x[i] <- 2
   }
}
x
 [1] 0 0 0 1 1 1 1 2 2 2

x <- 1:10
cut(x, c(1, 3, 7, 10), labels=F,include.low=T) - 1

ifelse(x < 4, 0, ifelse(x >= 8, 2, 1))
```

```
x <- cbind(a=1:5, b=5:1, c=1:5)
x
     a b c
[1,] 1 5 1
[2,] 2 4 2
[3,] 3 3 3
[4,] 4 2 4
[5,] 5 1 5

apply(x, 1, sum)
[1]  7  8  9 10 11

apply(x, 2, sum)
 a  b  c
15 15 15
```

Apply your functions

Apply more

tapply (aggregate)

lapply, sapply

```
x <- list(1:10)

lapply(x, function(x) x + 5)
[[1]]
 [1]  6  7  8  9 10 11 12 13 14 15
```

Functionality

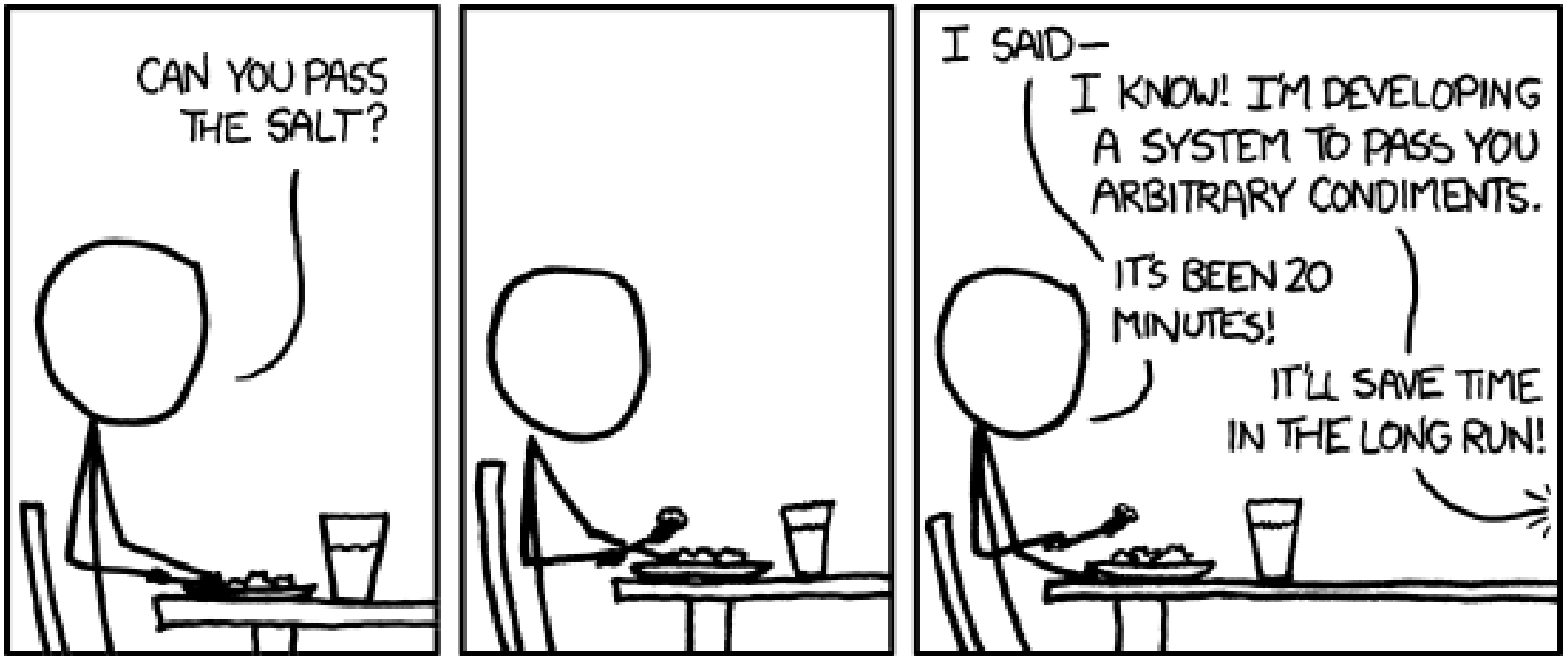
```
f <- function() {
    print("Mangwanani akanaka!")
}
```

```
 f()
 [1] "Mangwanani akanaka!"
 > f()
 [1] "Mangwanani akanaka!"
 > f()
 [1] "Mangwanani akanaka!"
```

Functionality

```
doit <- function(a, b) {
   x <- a + b
   return(x)
}

> doit(1, 2)
[1] 3
> doit(1:10, 2)
 [1]  3  4  5  6  7  8  9 10 11 12
>
```


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

tidyverse

```
a <- 1:10
b <- sum(a)
d <- sqrt(b)
e <- log(d)

e <- log(sqrt(sum(a)))

e <- a |> sum() |> sqrt() |> log()

library(magrittr)
e <- a %>% sum %>% sqrt %>% log
```

Read and write

```
x <- read.csv("c:/data/observations.csv")
readxl::read_excel ("c:/data/observations.xlsx")

foreign::read.dbf
foreign::read.dta        for Stata files
foreign::read.spss       for SPSS files
foreign::read.ssd        for SAS files

write.csv(x, file="c:/data/results.csv")

saveRDS(x, "file.rds")
y <- readRDS("file.rds")
```

Before you start

- …
- …
- …
- …
- …
- …