Welcome to the R workshop!

Please download all of the files in: http://tiny.cc/gwzft

Doug Jackson and Dave Allen

August 31, 2011

An Introduction to R

Doug Jackson and Dave Allen

August 31, 2011

What is R

- R is an open-source, free statistical and graphing software package
- it is built on the S computer language

What is R

- R is an open-source, free statistical and graphing software package
- it is built on the S computer language

What can R do? *

- effective data handling and storage facility
- calculation on arrays and matrices
- has a collection of tools for data analysis
- produce graphics for display on computer or hardcopy
- programming language (S) including conditionals, loops, functions, …

* List from http://cran.r-project.org/doc/manuals/R-intro.html#The-R-environment

Why use R?

- Functionality of a wide range of other programs (data storage, graphics, statistics, programming)

- Free, open source, and available on almost all platforms (Mac OSX, Unix, Windows, Linux)

- extensive packages (CRAN has packages for maximum likelihood parameter estimation, GIS, phylogenic analysis, PCA, regression trees,…)

Why use R?

- Functionality of a wide range of other programs (data storage, graphics, statistics, programming)
- Free, open source, and available on almost all platforms (Mac OSX, Unix, Windows, Linux)
- extensive packages (CRAN has packages for maximum likelihood parameter estimation, GIS, phylogenic analysis, PCA, regression trees,…)

What is the catch?

- poor GUI
- buggy
- harder to do simple tasks than in excel, or even SAS or SPSS
- not as efficient at some tasks as lower-level languages (like C or FORTRAN)

Not a panacea, you need to evaluate your needs and find best tool for you

Philosophy and outline of the workshop

Doug Jackson and Dave Allen
31-Aug-2011

Philosophy

no prior experience with R

basic stats knowledge

no prior experience with programming

Philosophy: familiarity, not mastery

what is R?

when should I use it?

how do I get started?

how do I get help?

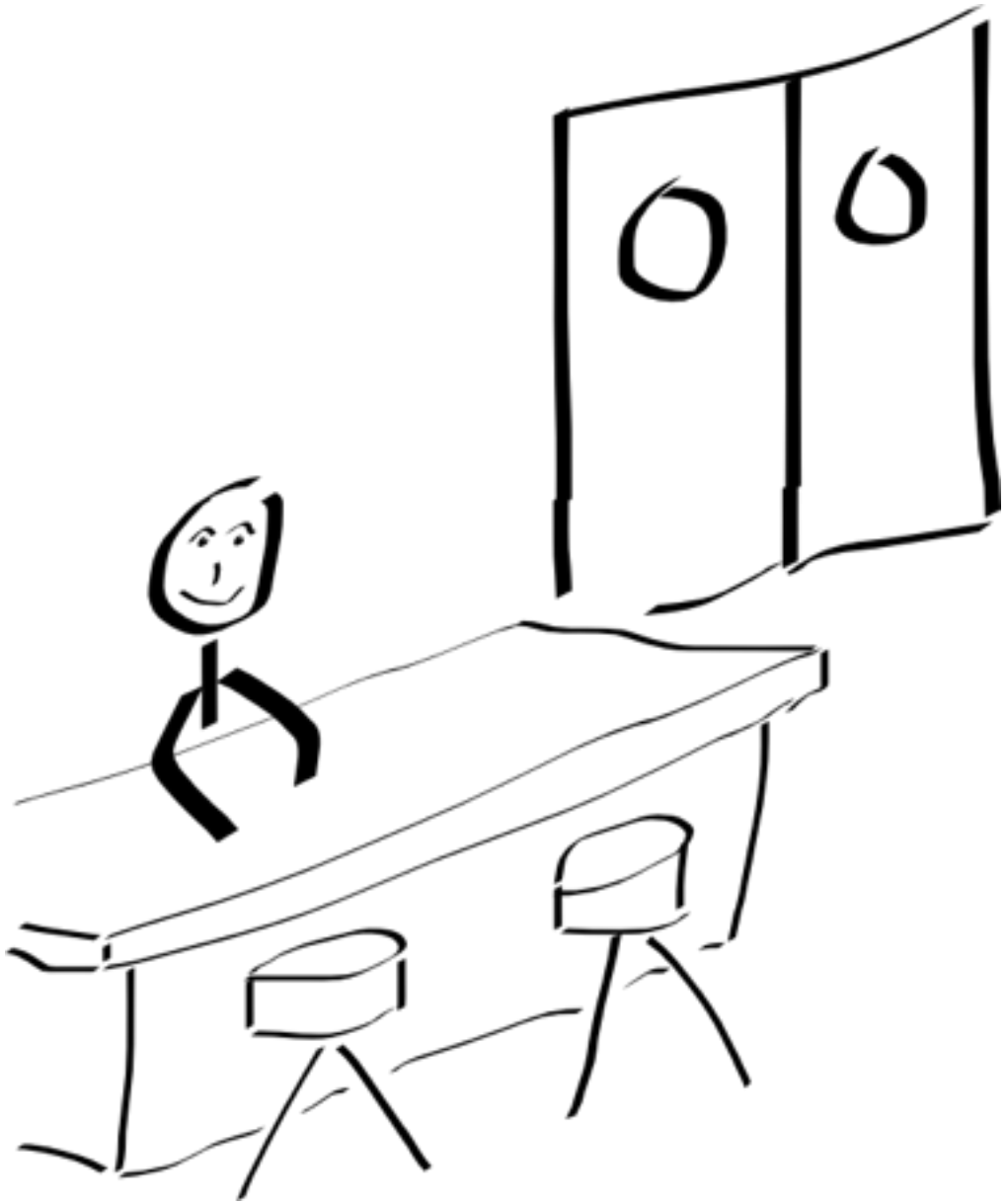
Outline

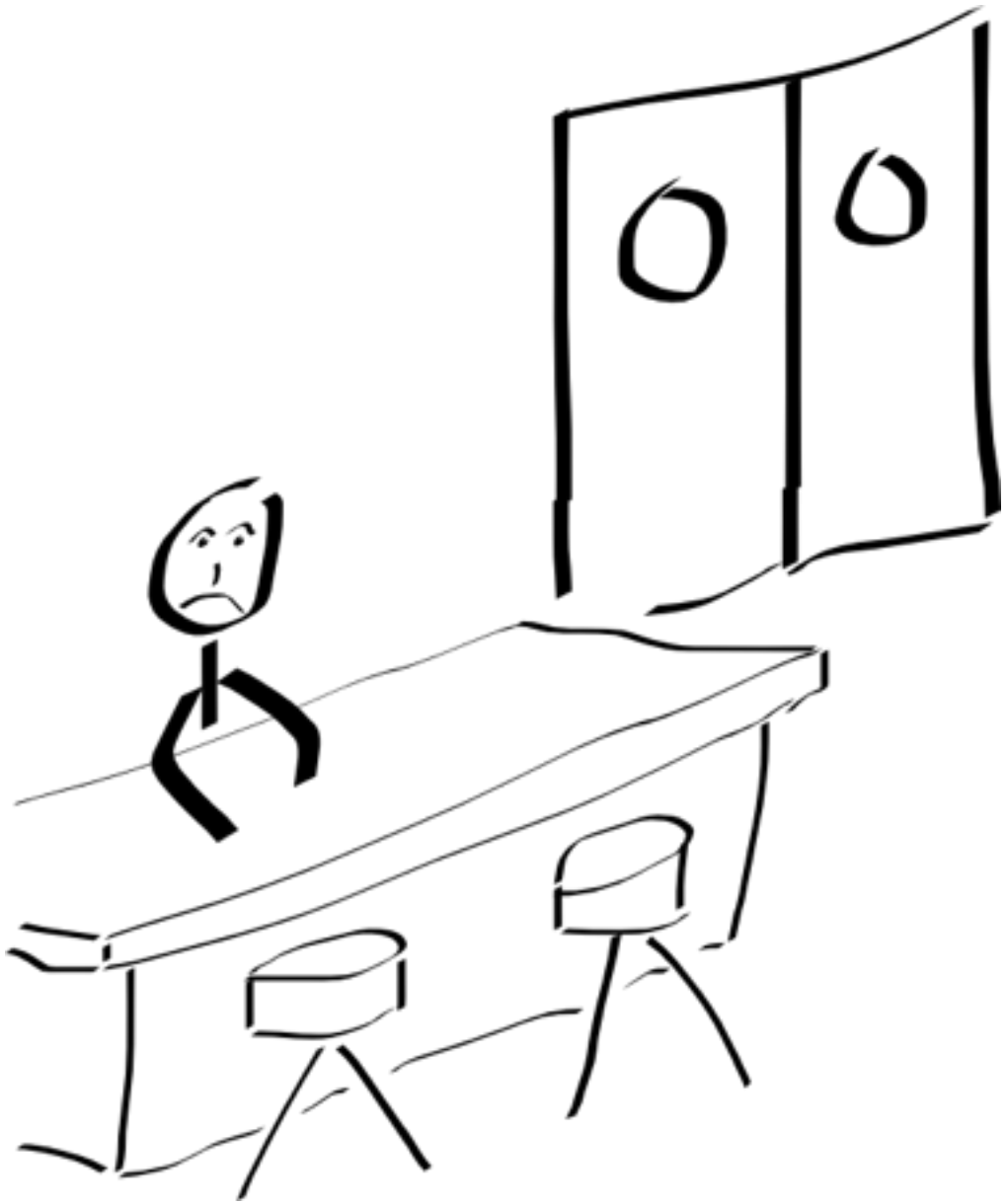
- Introduction to R
- Basic concepts: overview of the R environment
- Dealing with data
- Data analysis and graphing
- Programming
- Capstone project

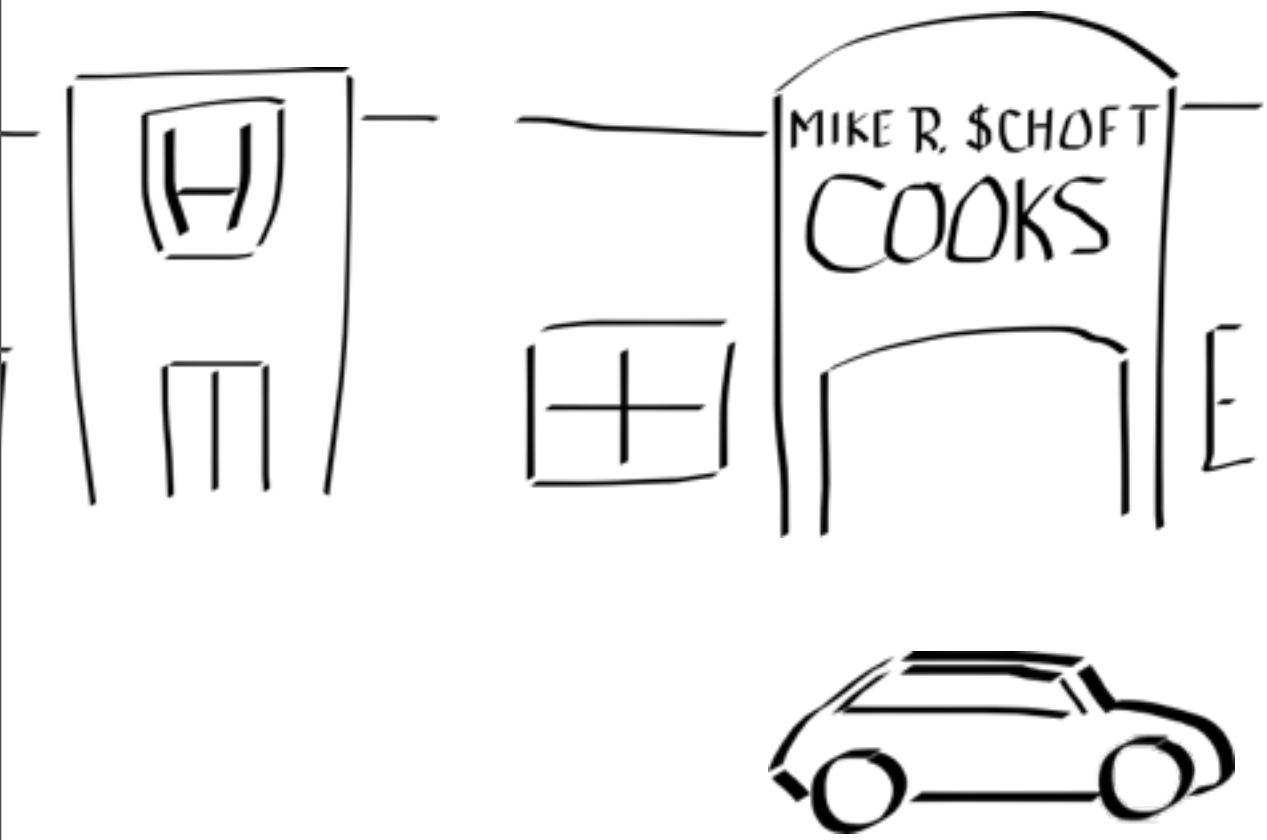
Basic concepts in R: Overview of the R environment

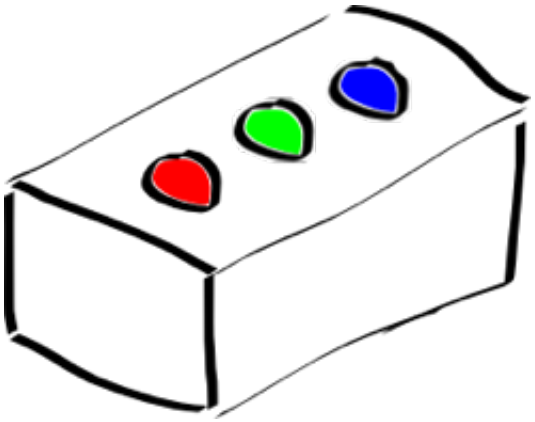
Doug Jackson and Dave Allen
31-Aug-2011

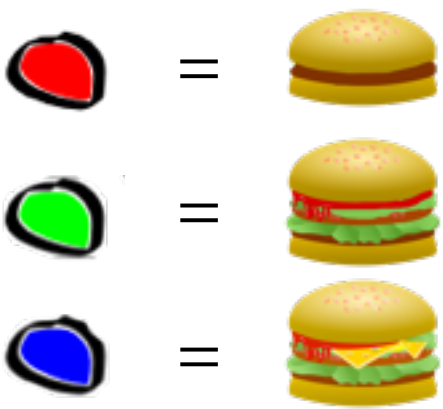

FEEL GOOD
DINER
Coca-Cola
Coca-Cola



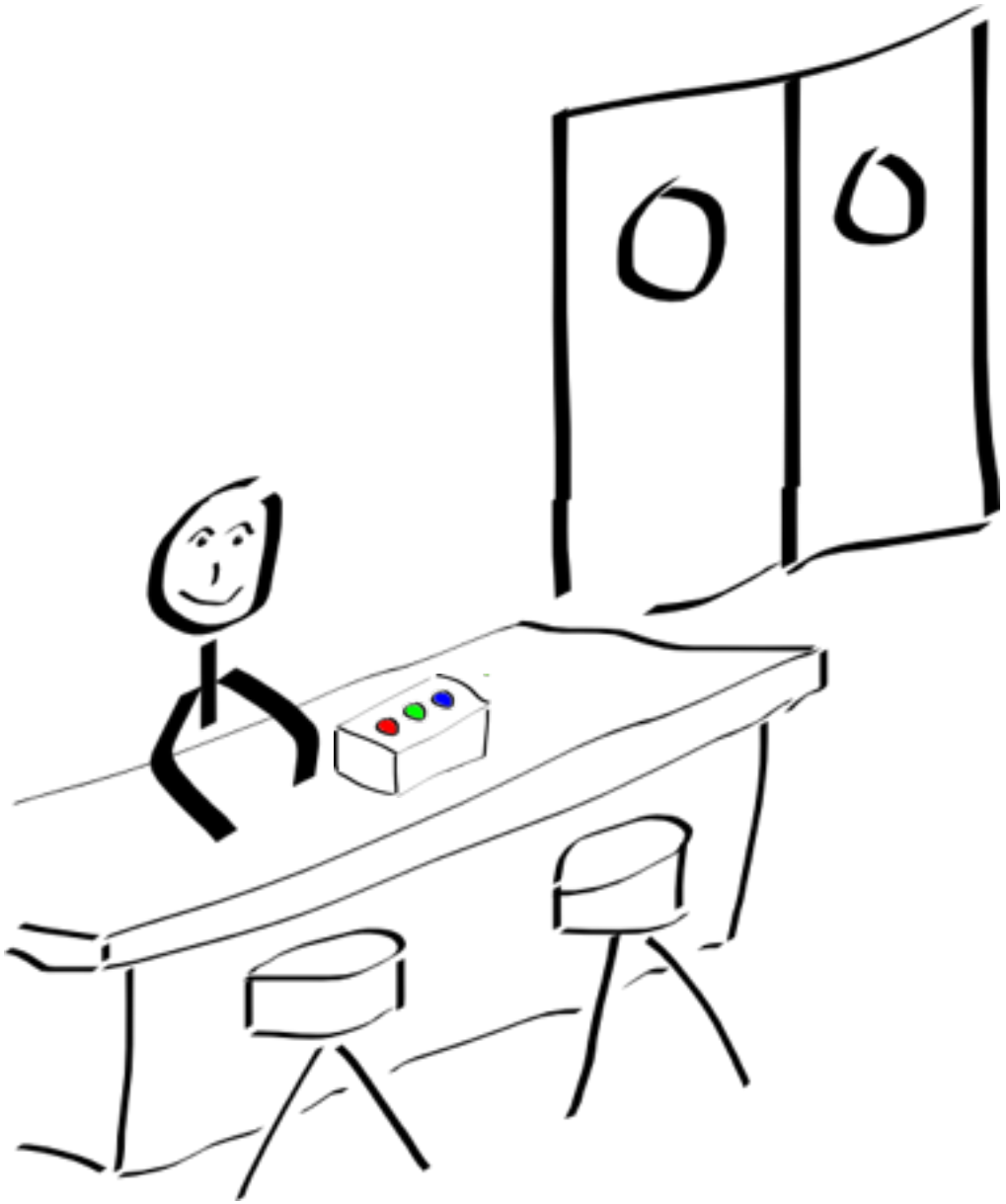


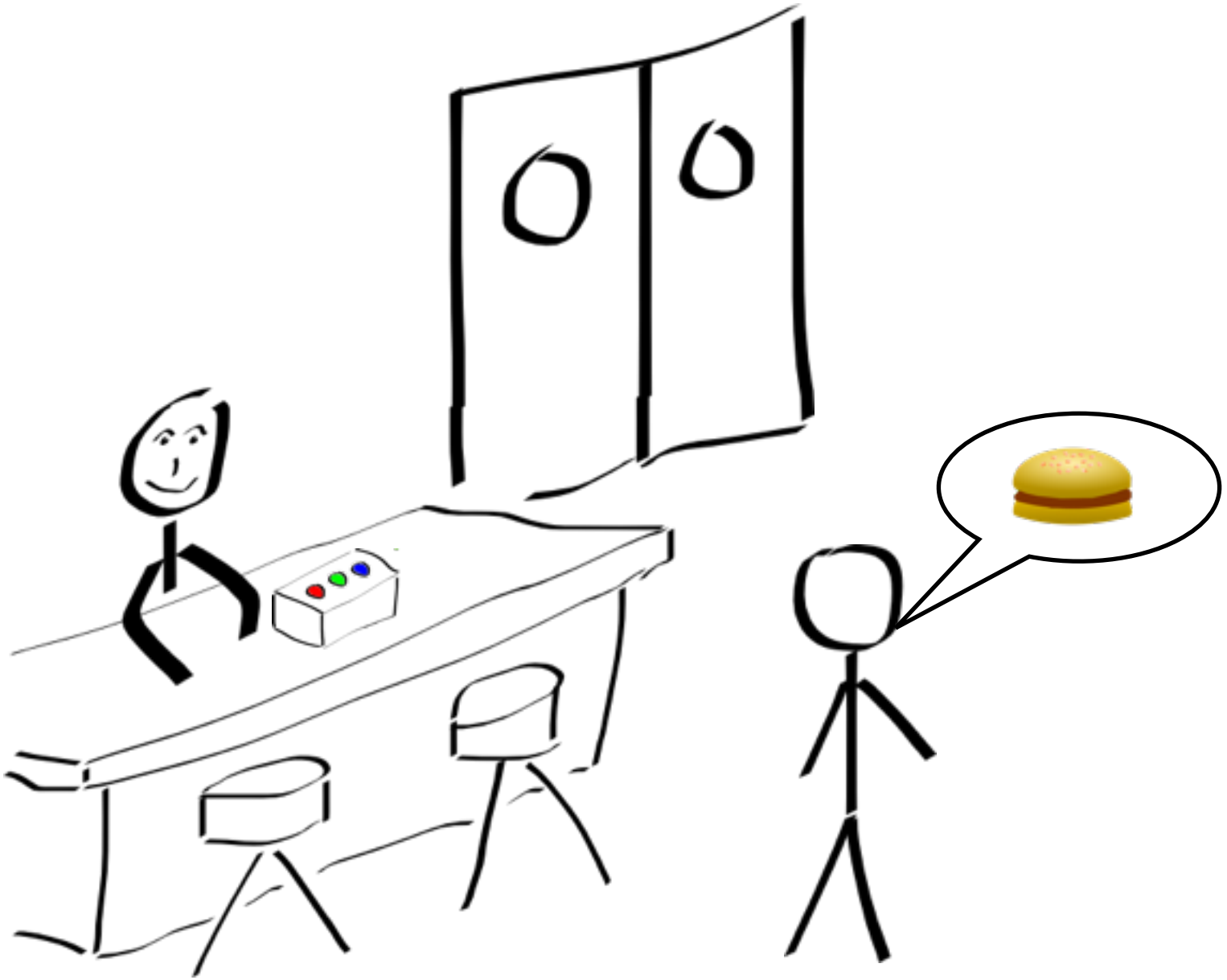

MIKE R. $CHOFT
COOKS

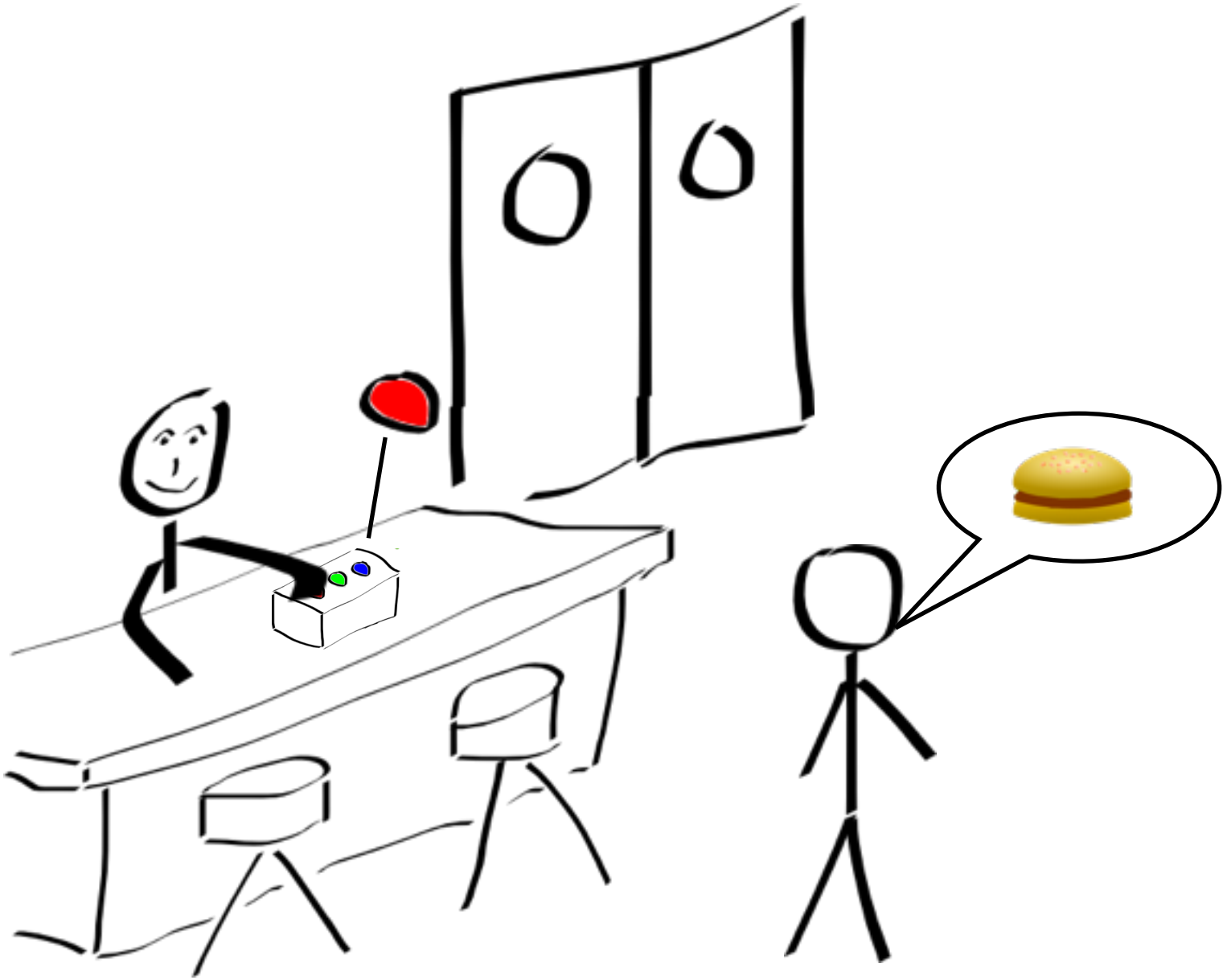


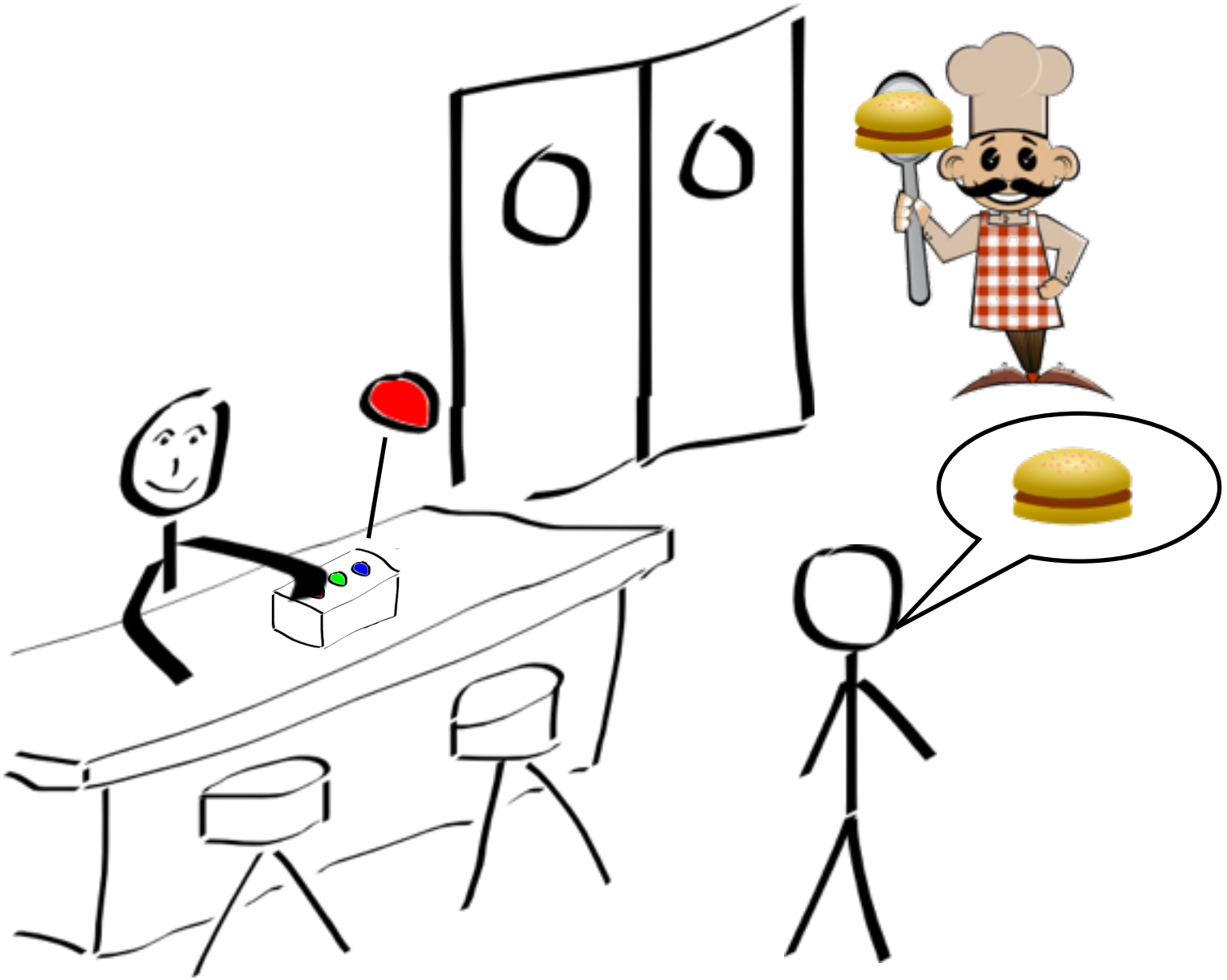


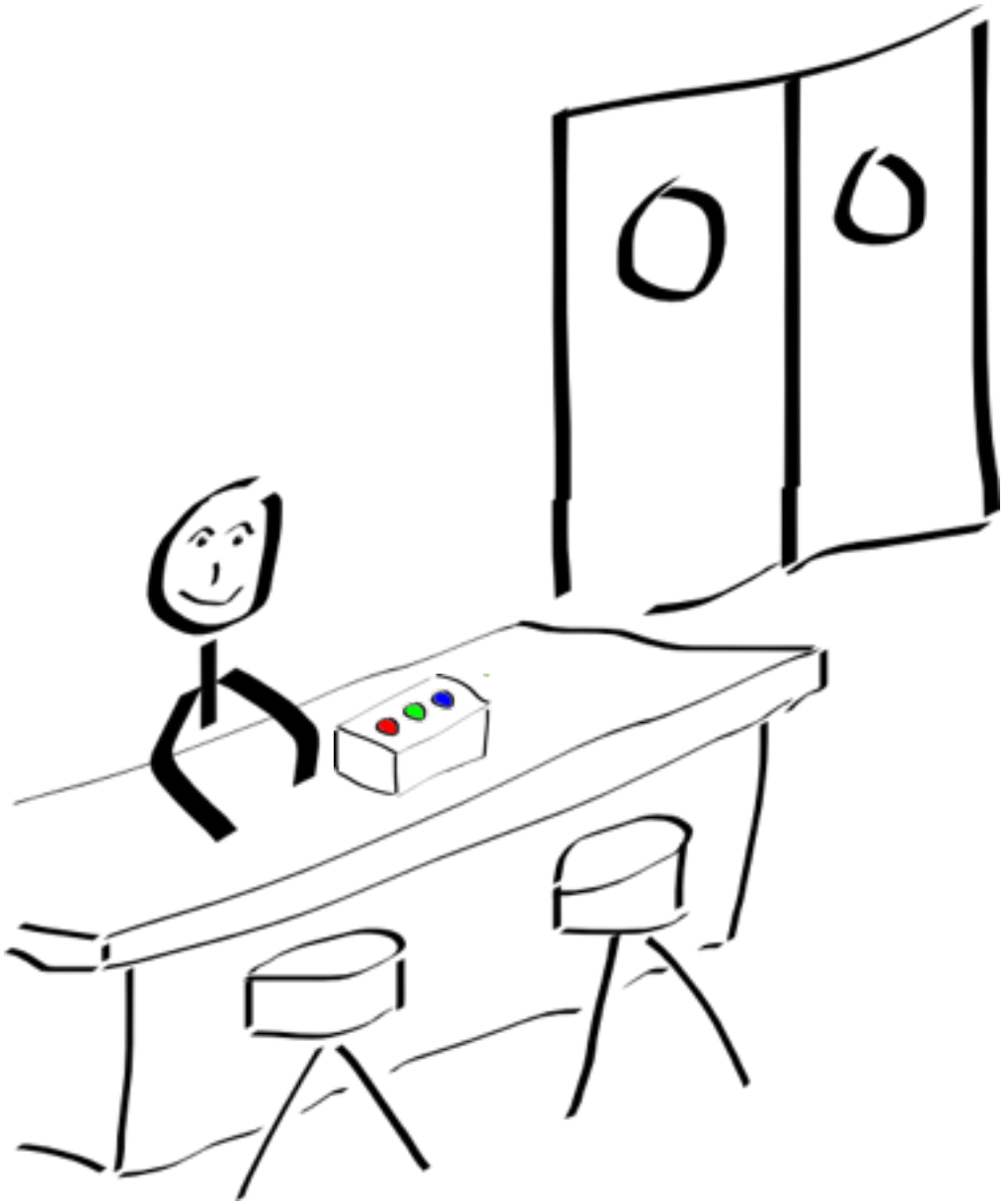

MIKE R. $CHOFT
COOKS

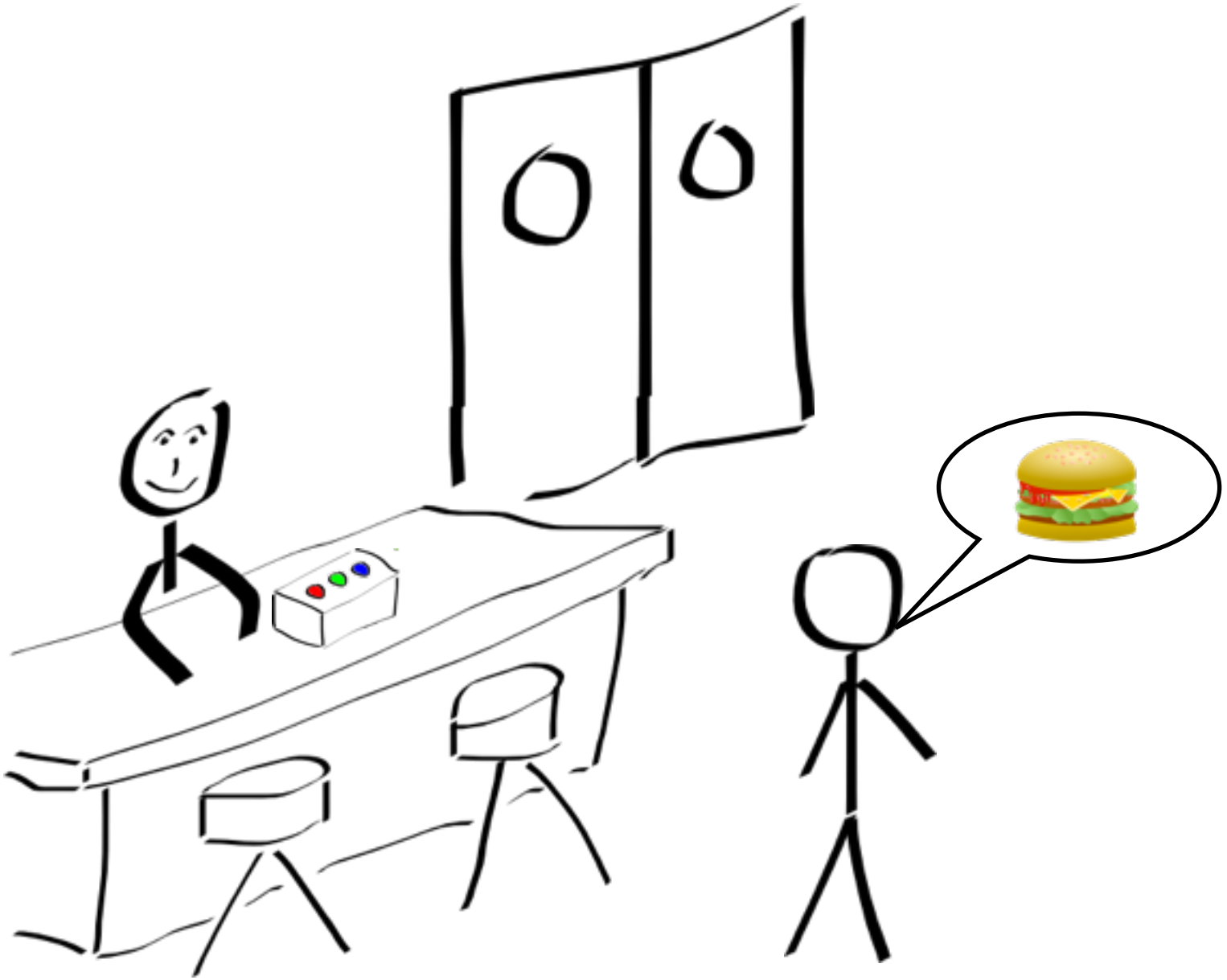


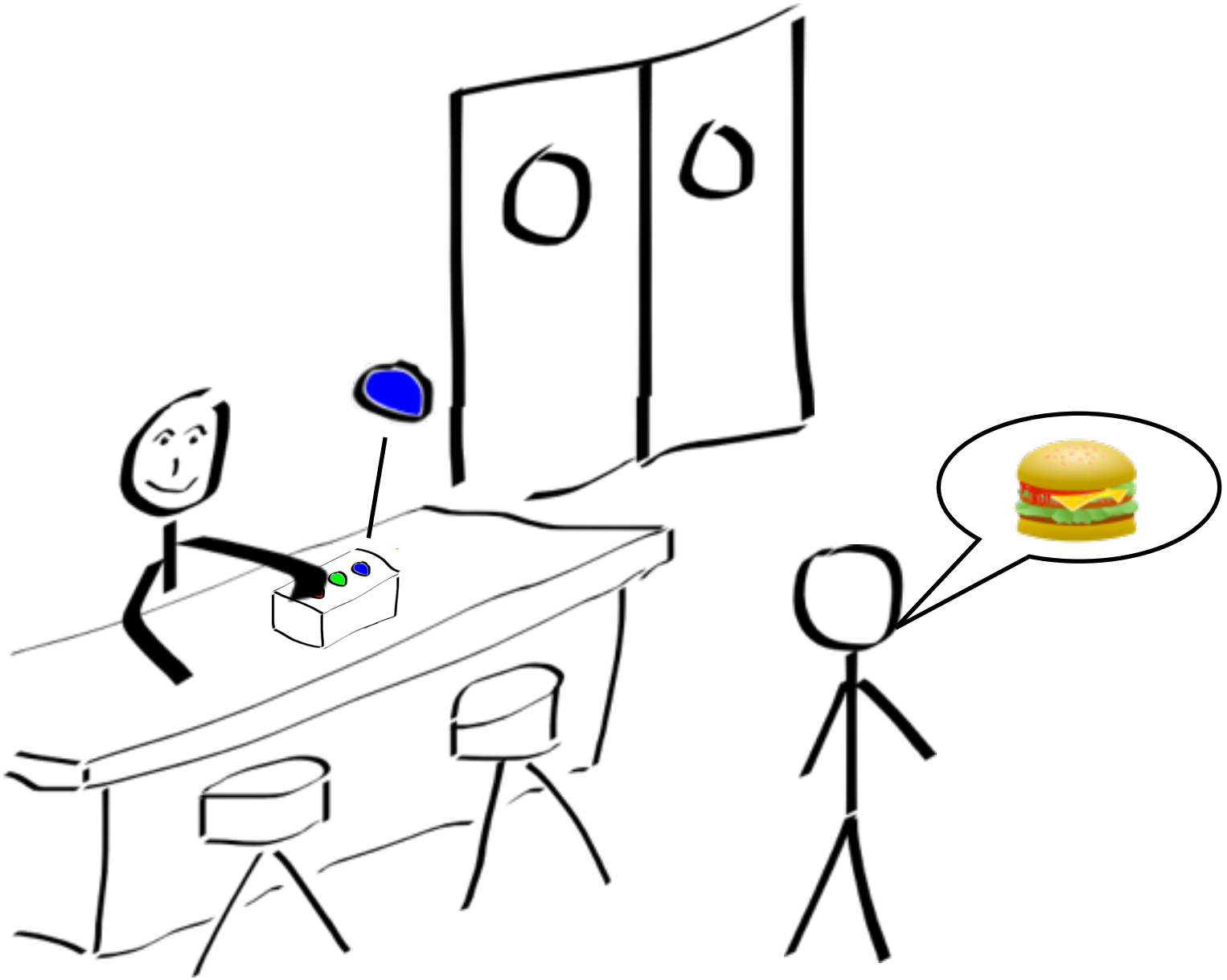


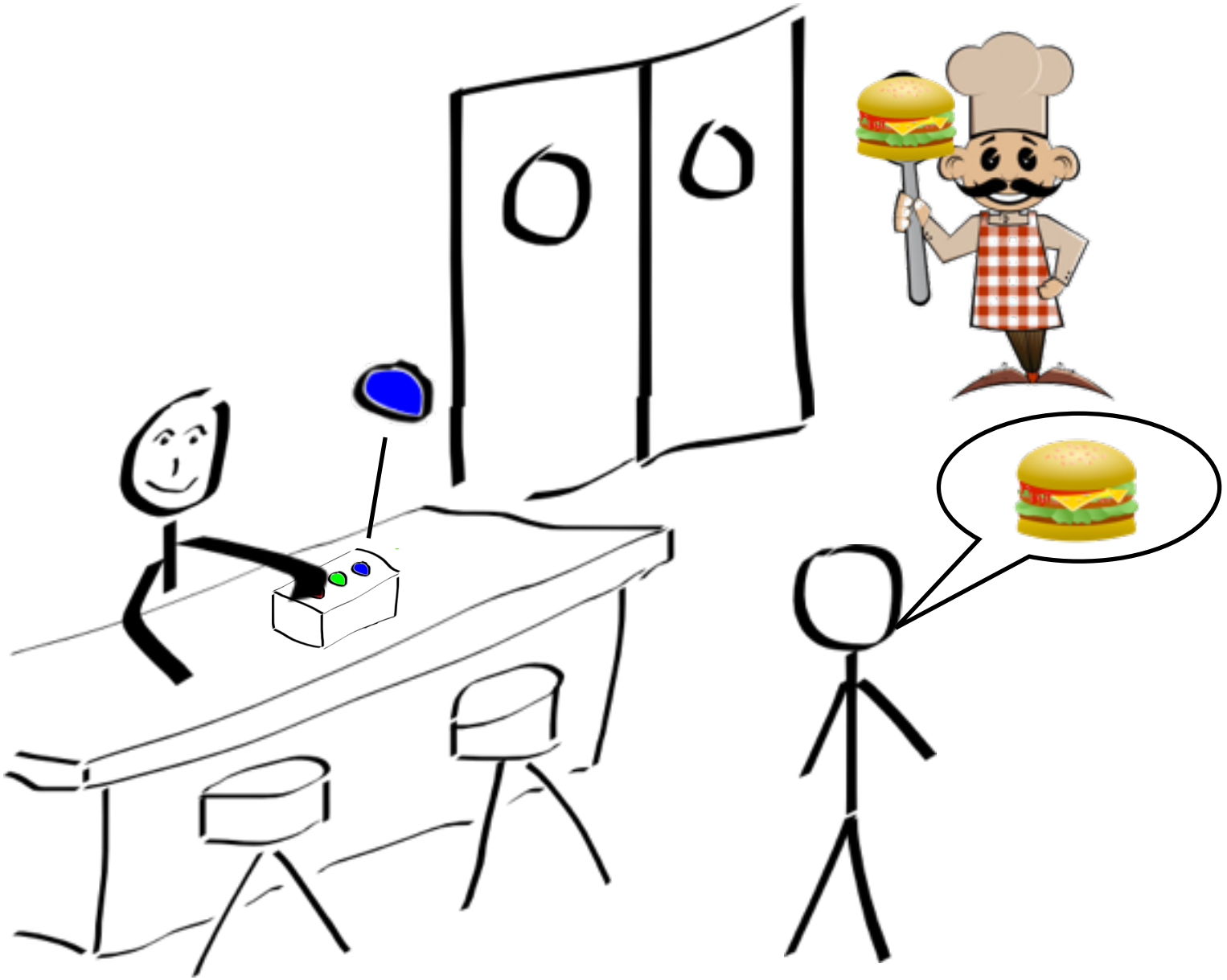


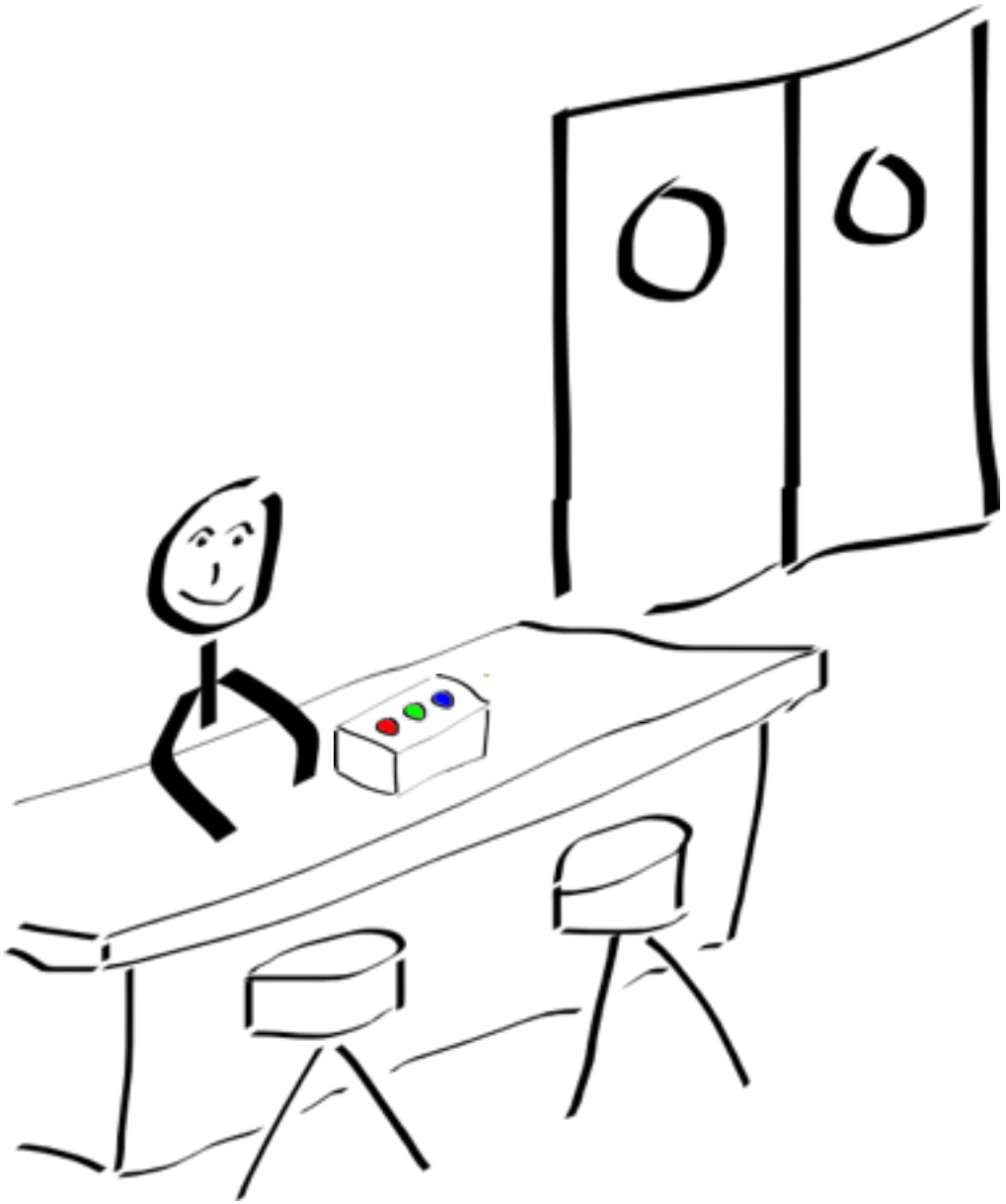


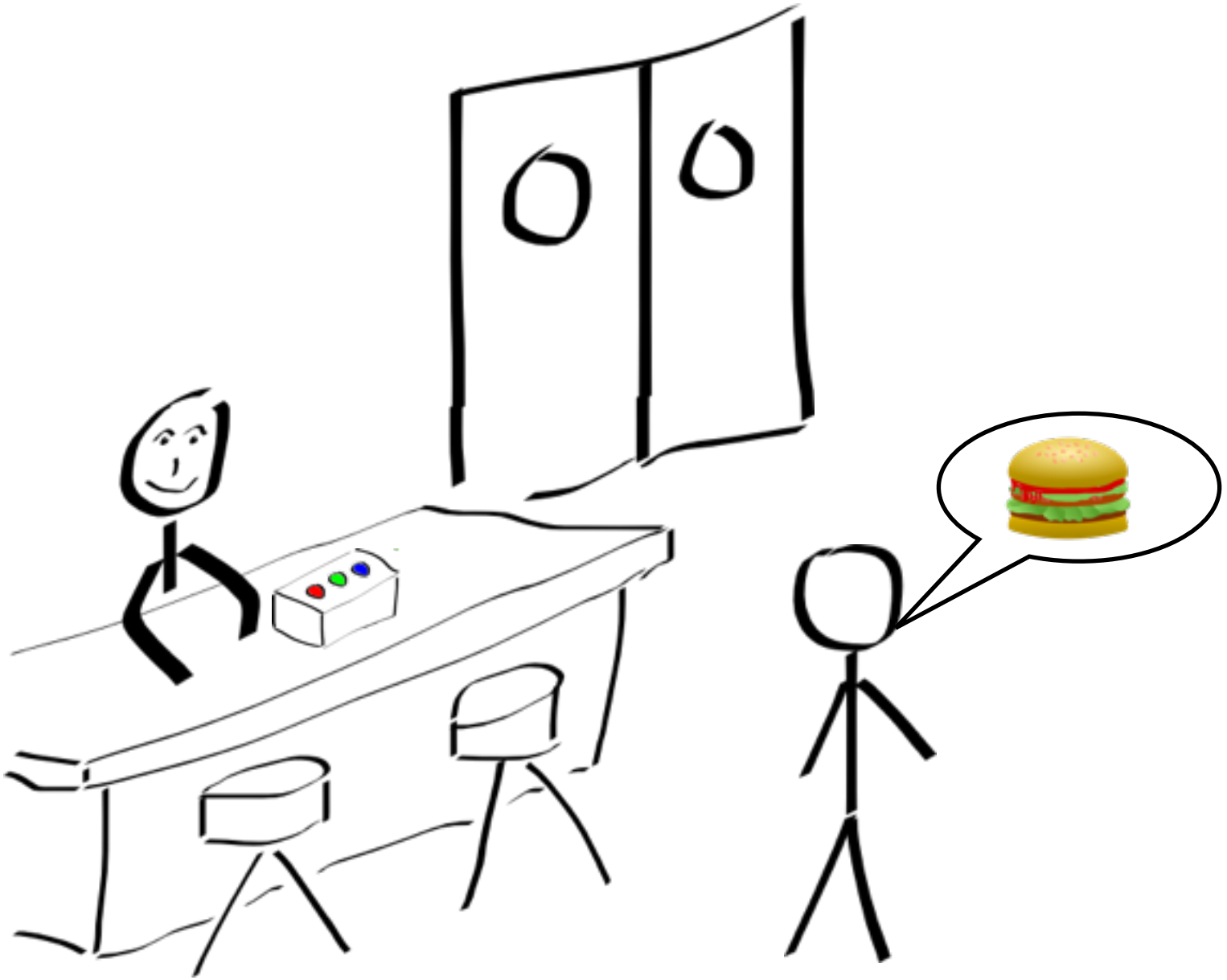


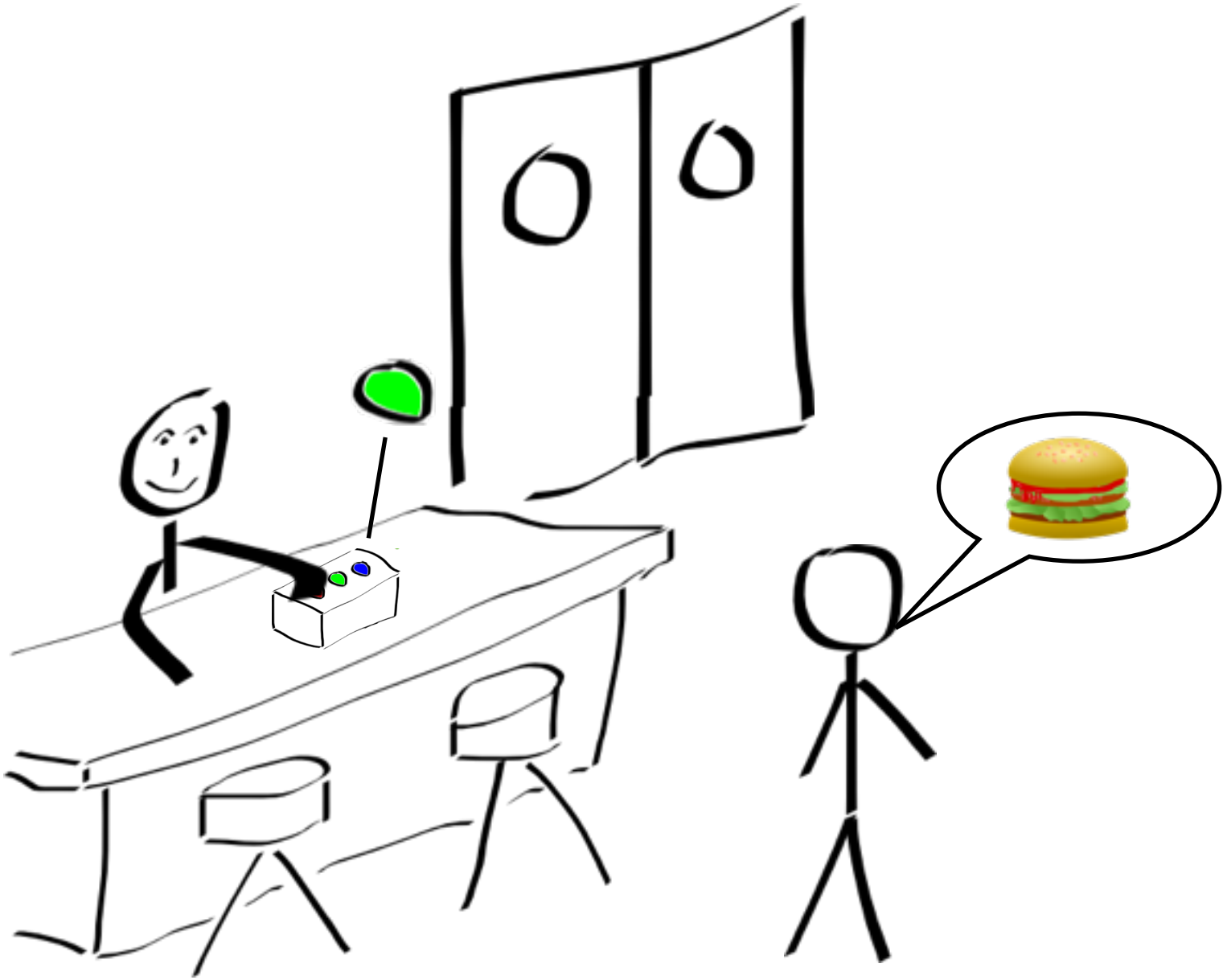


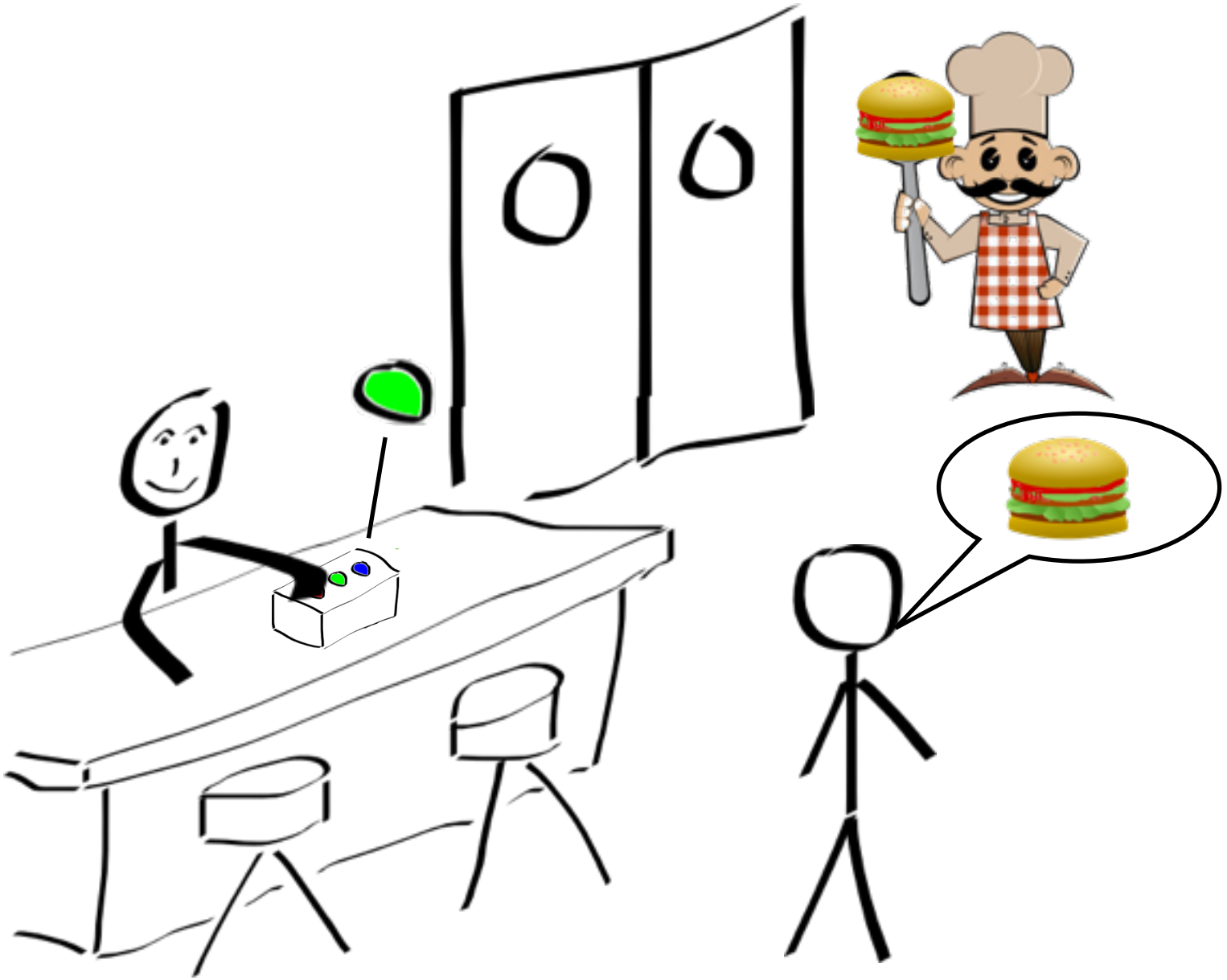


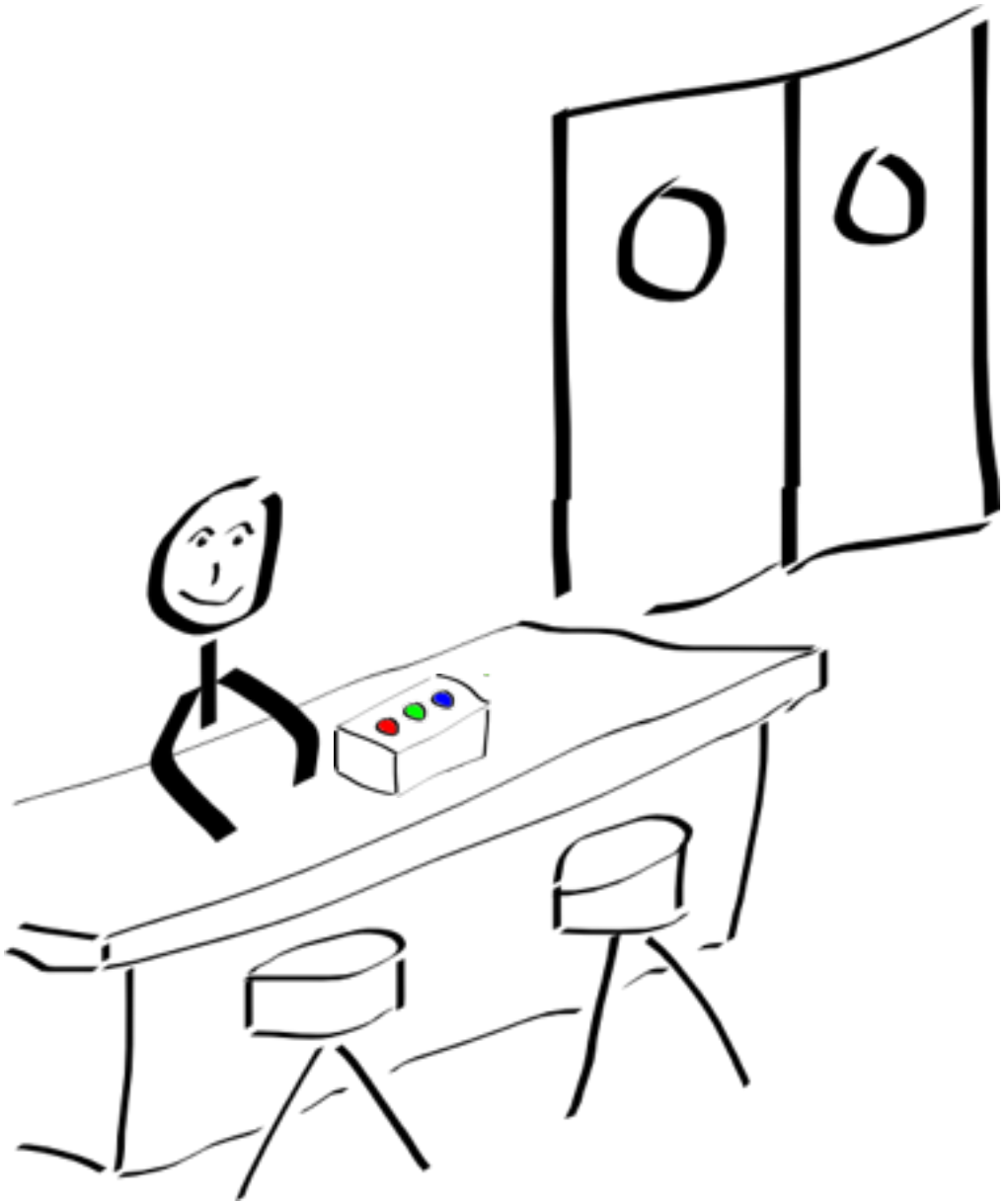


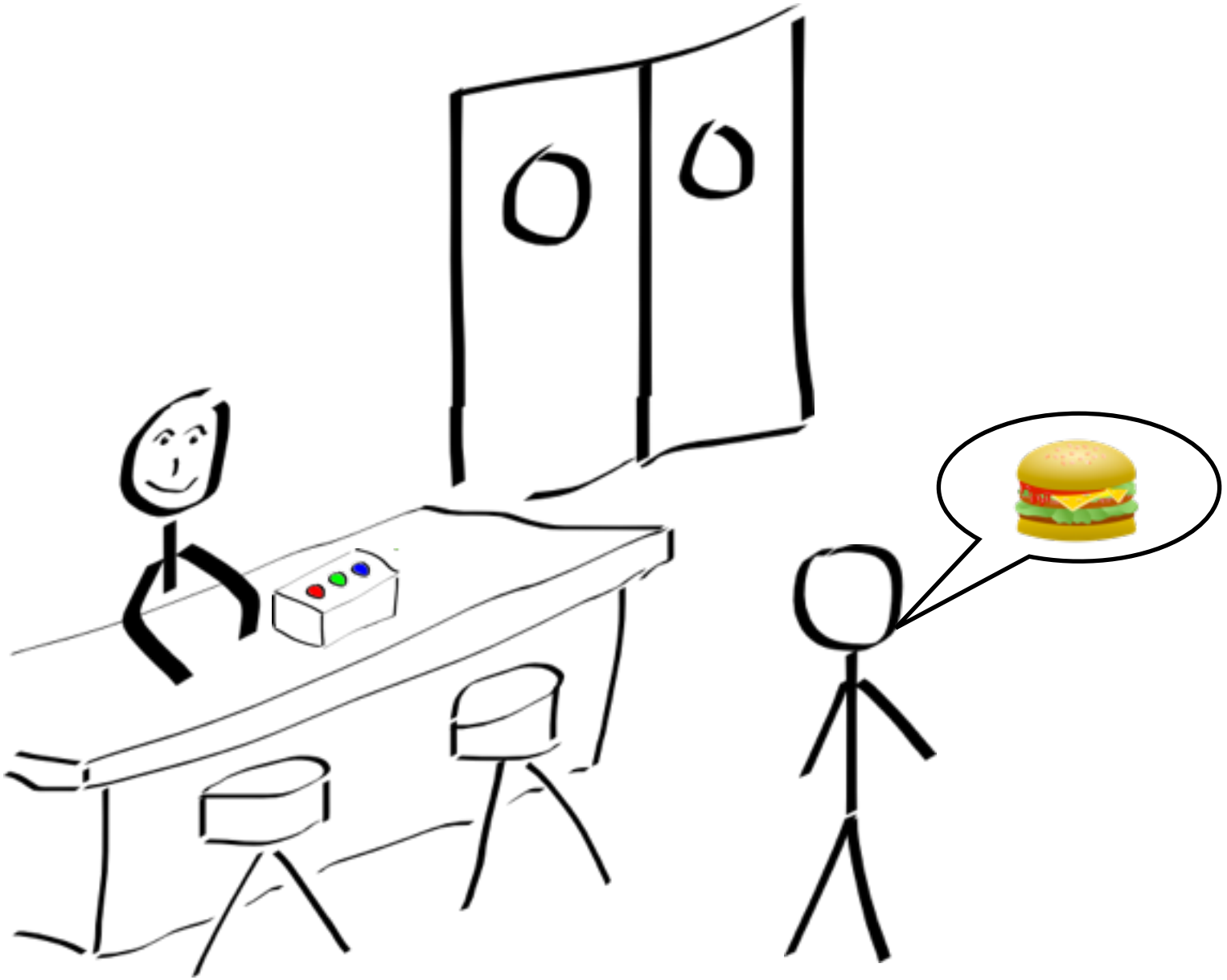


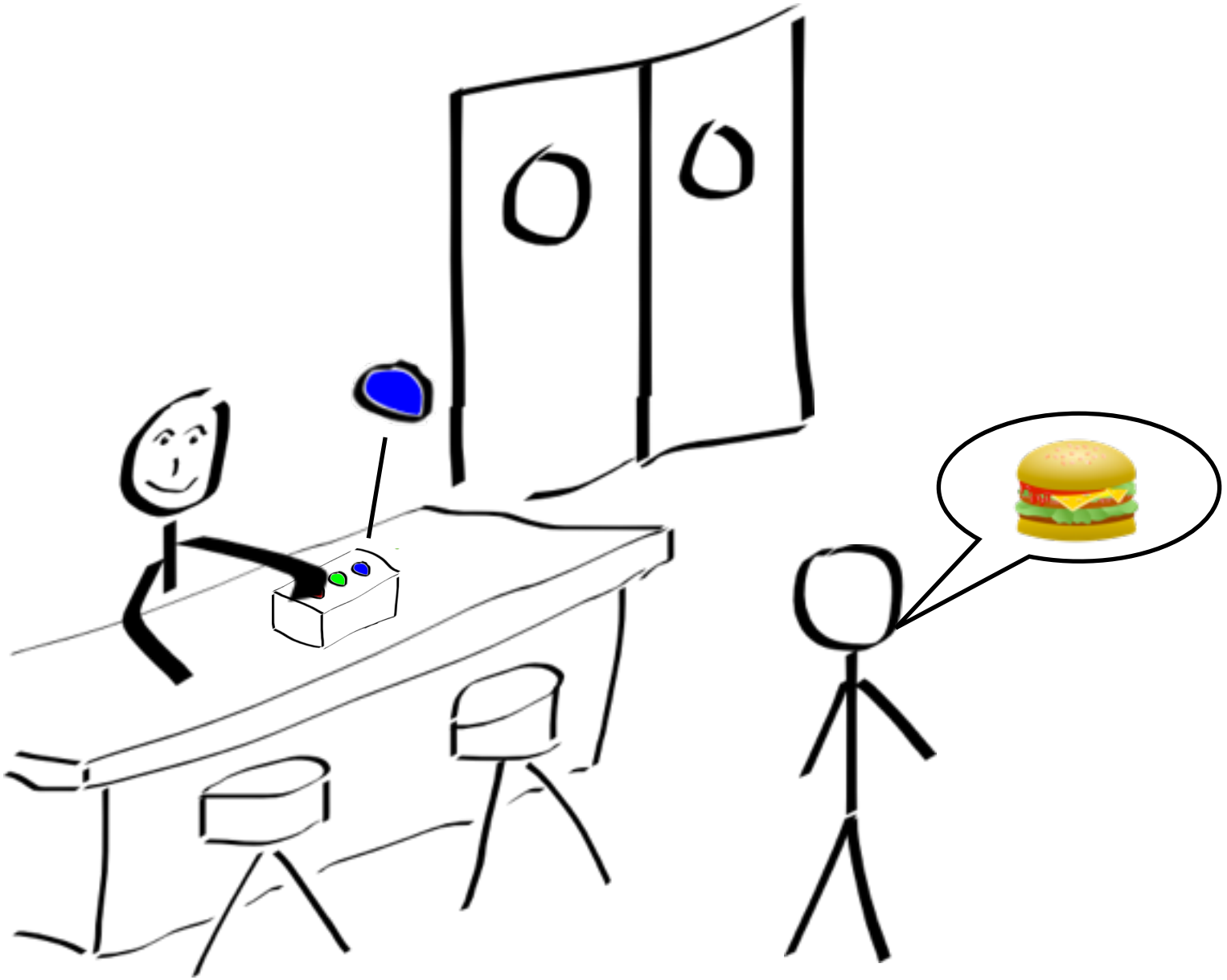


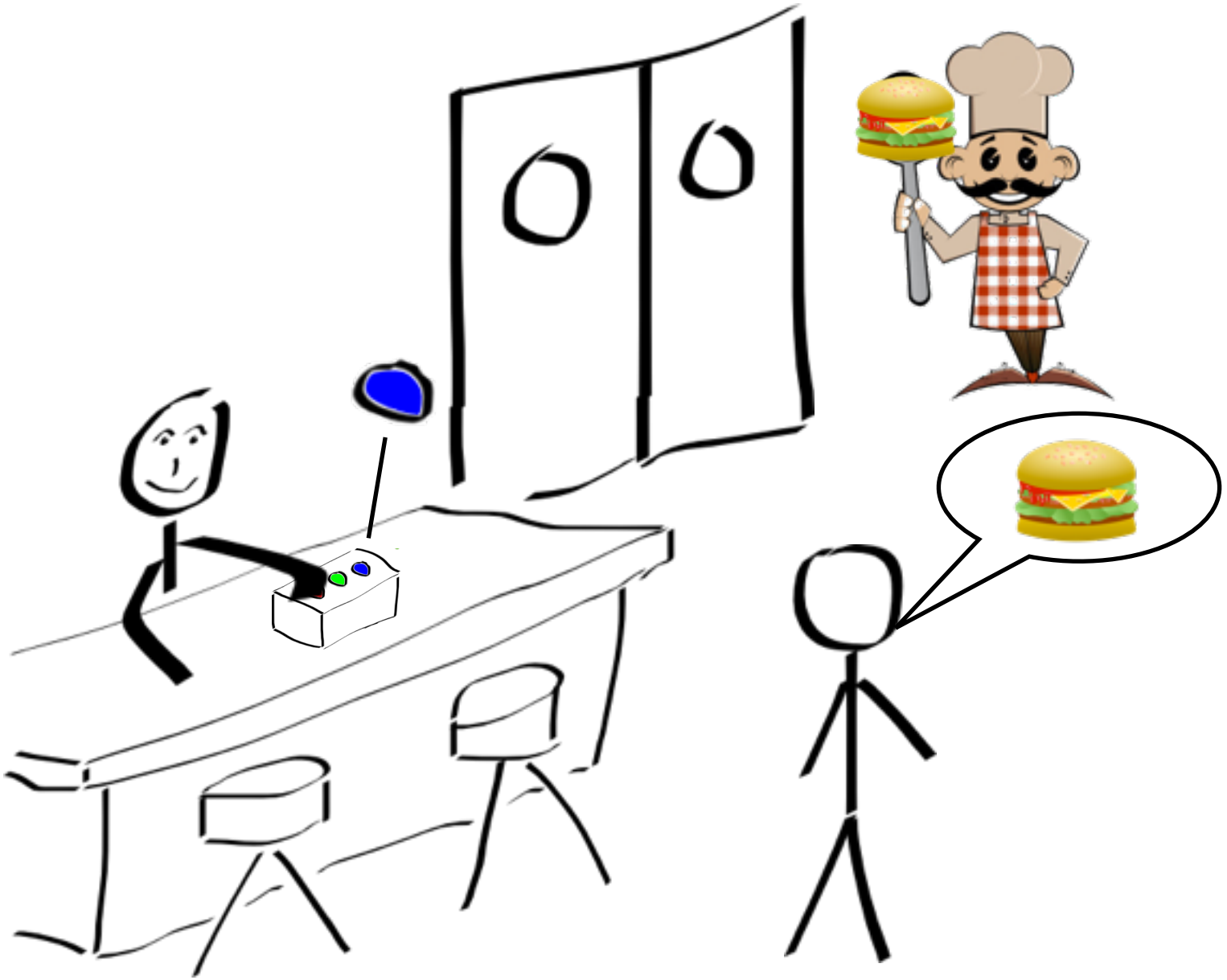


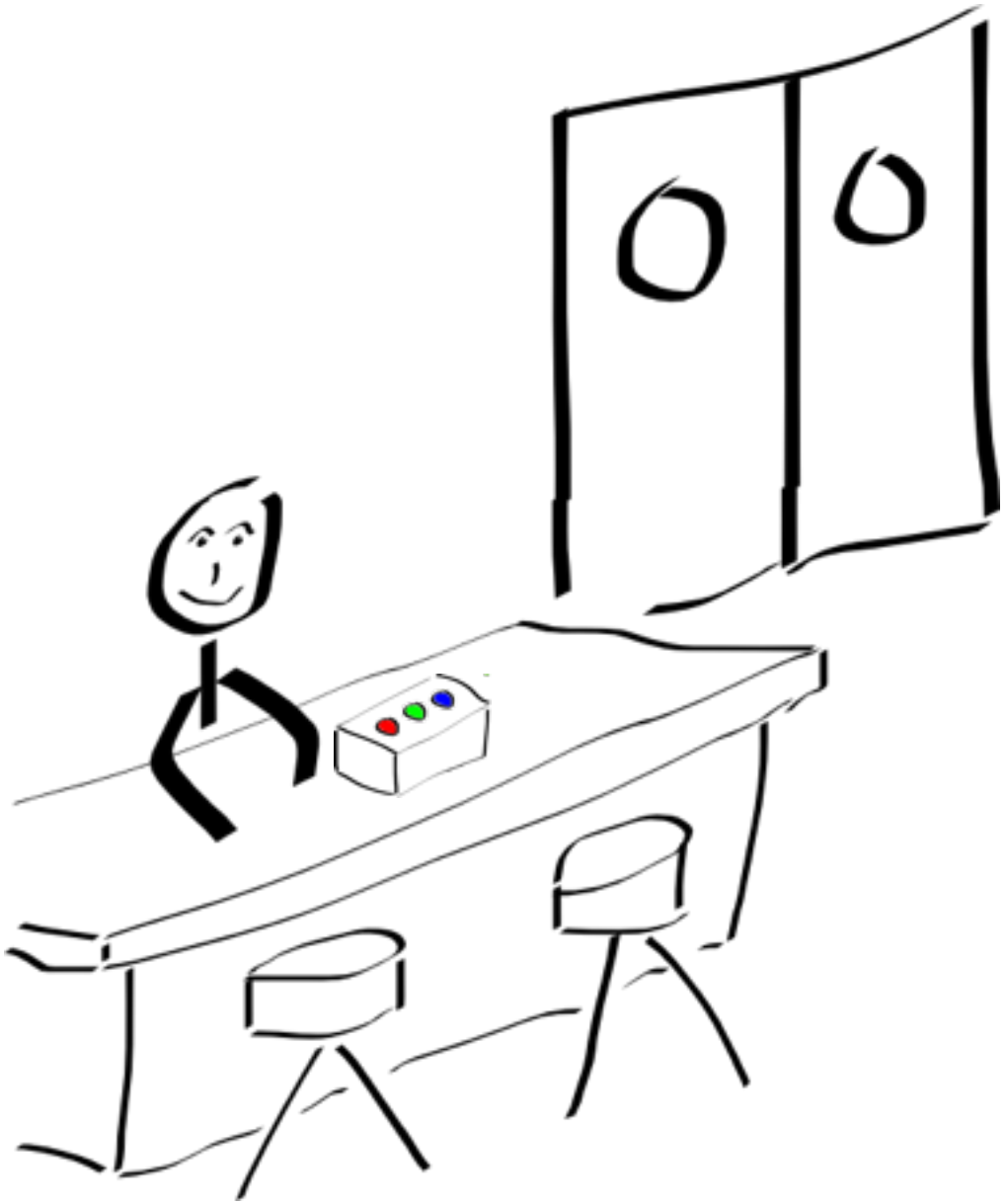


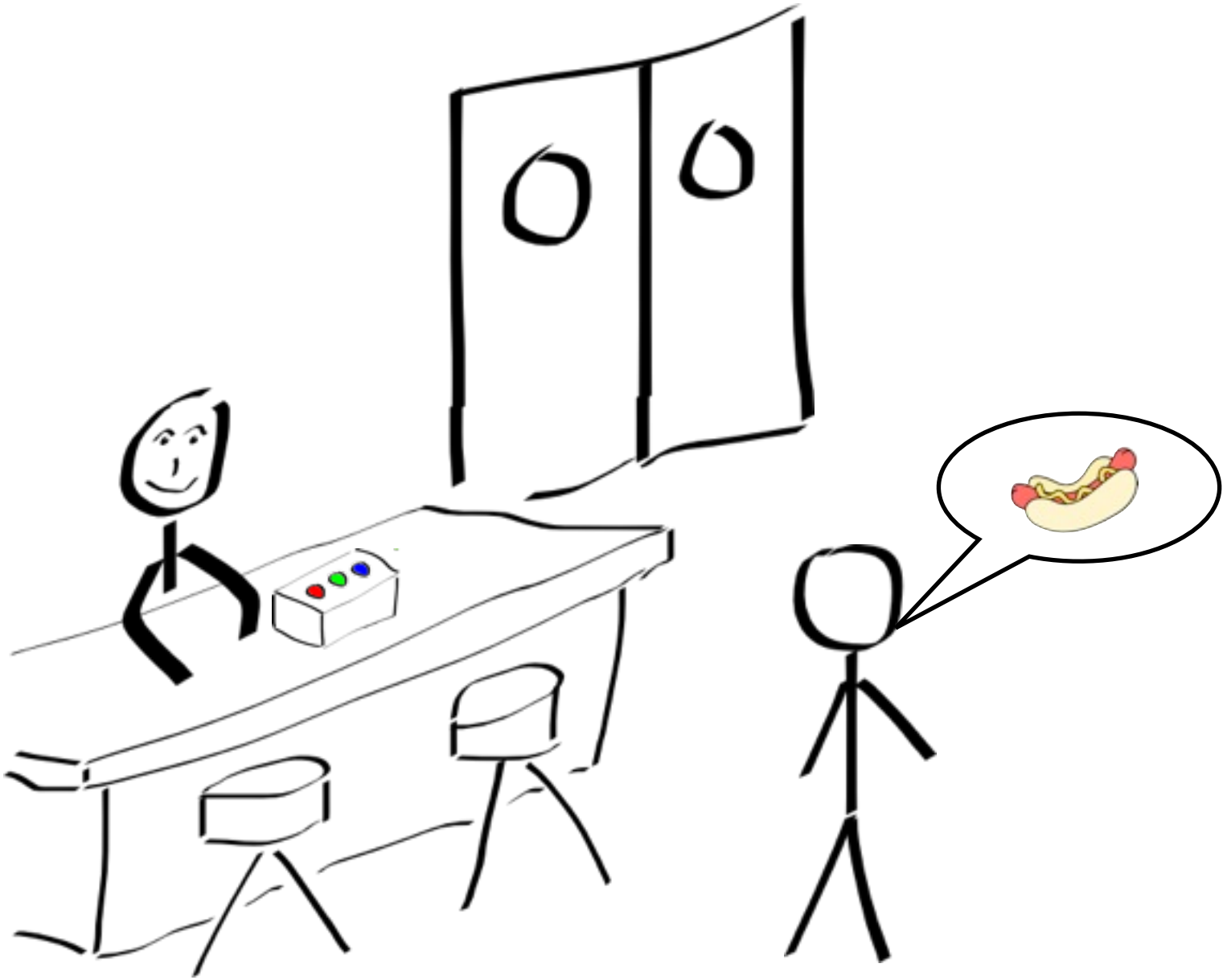


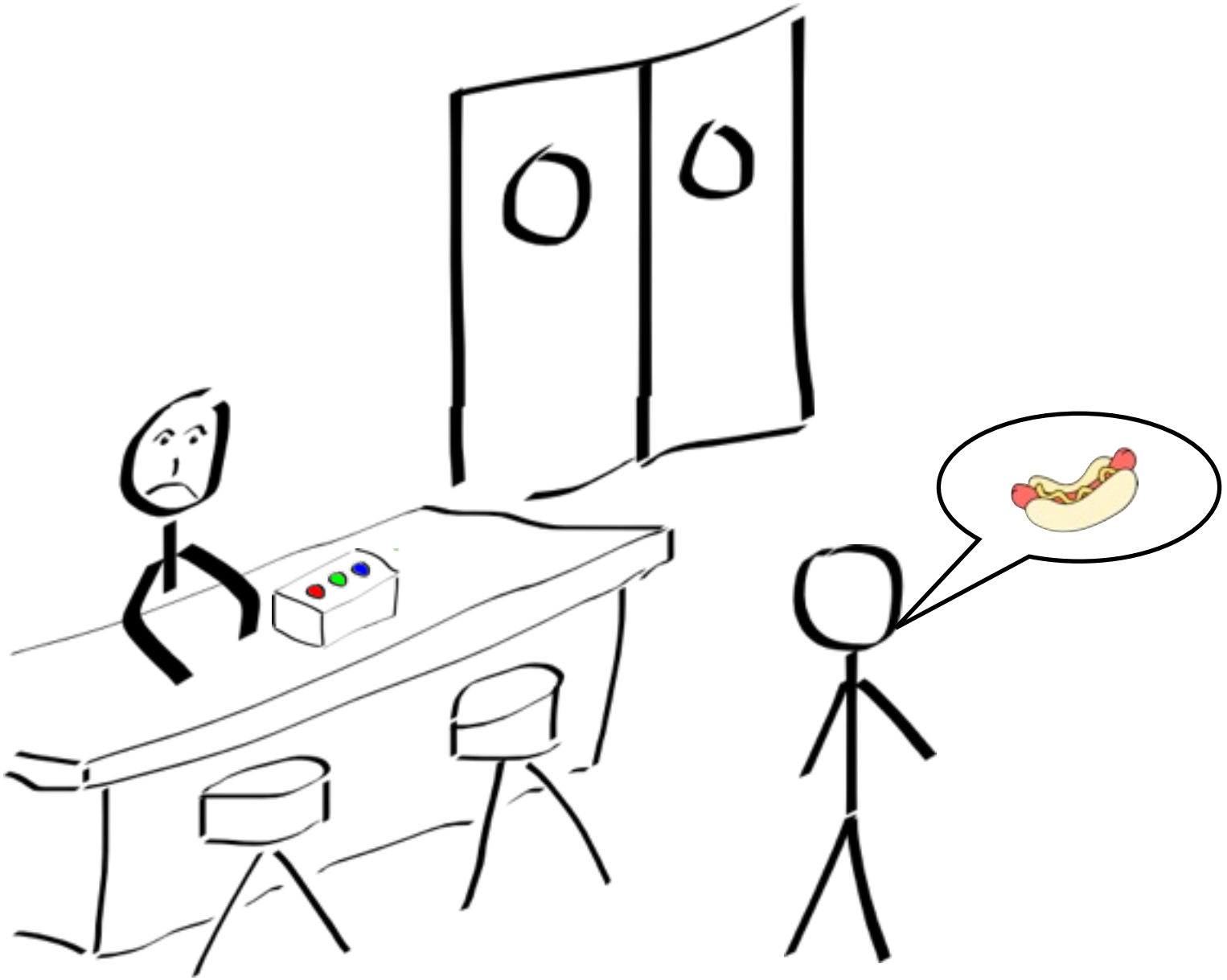


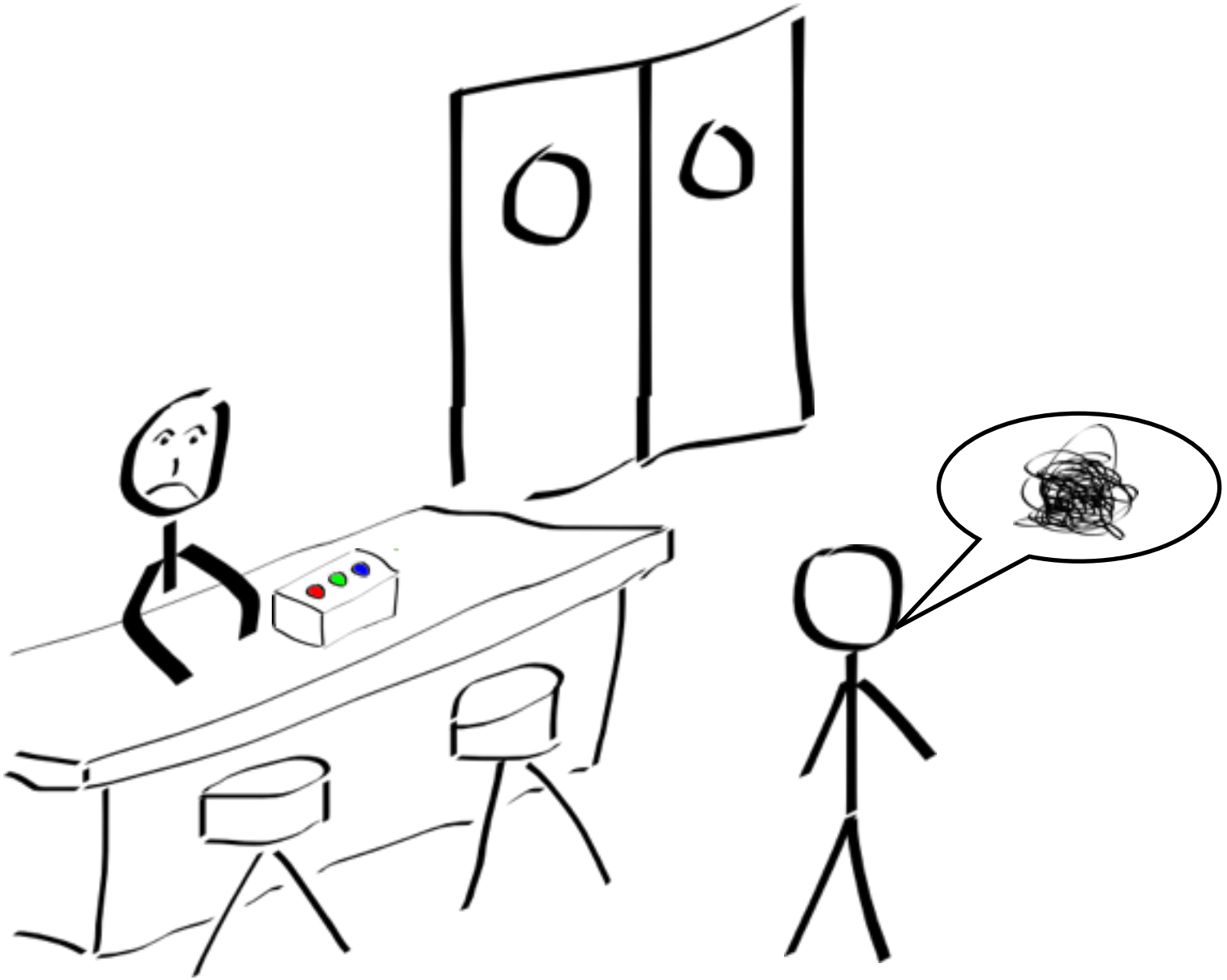






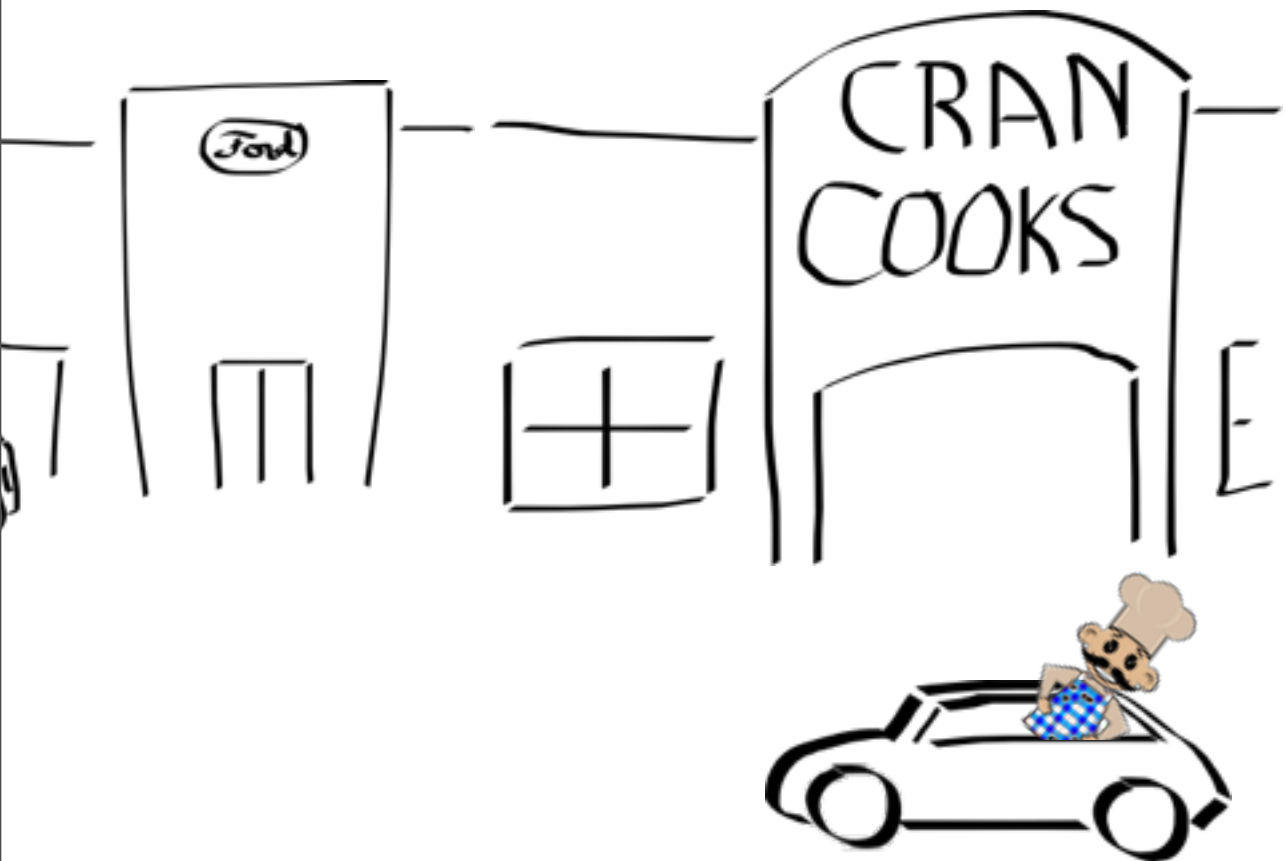


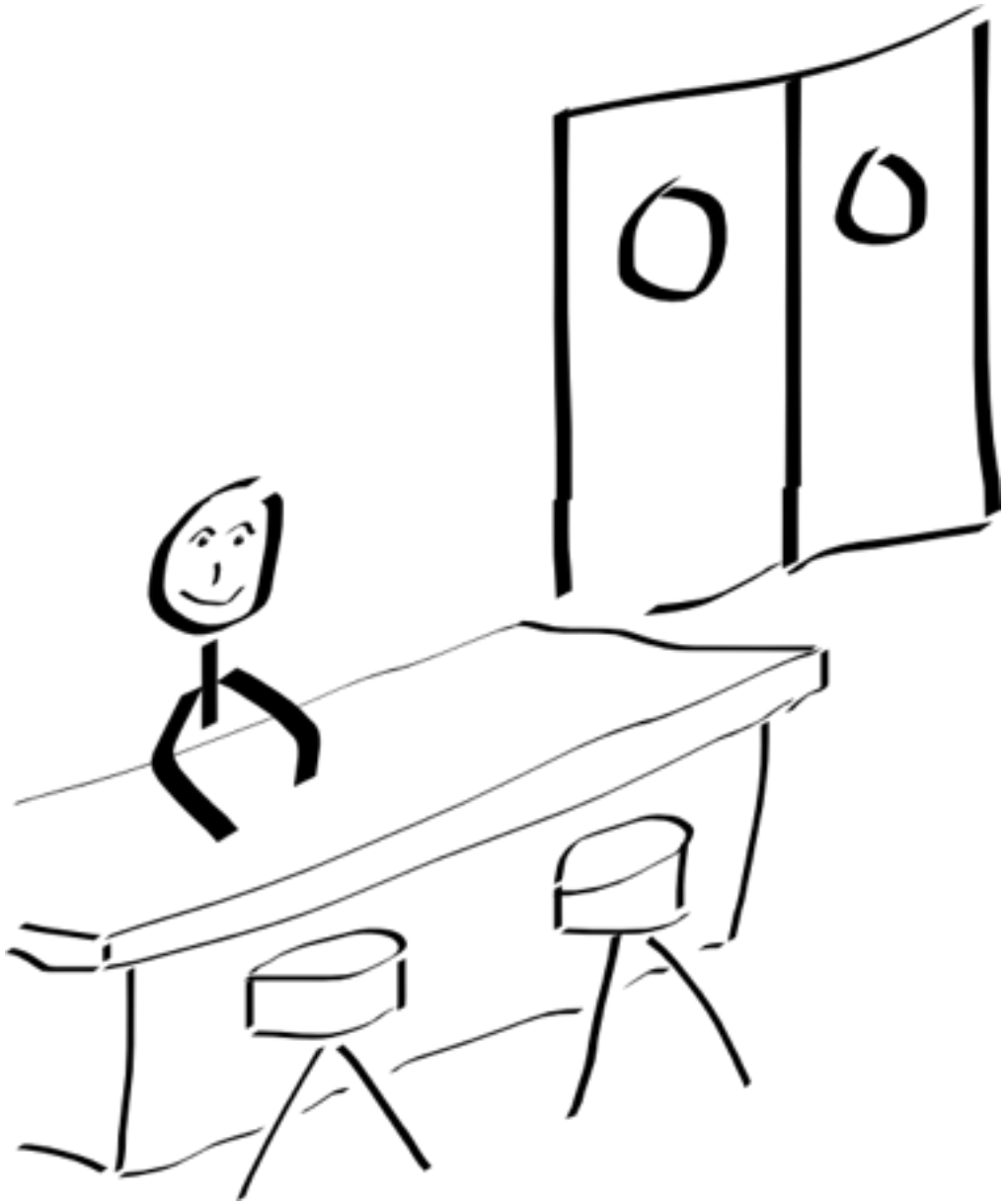


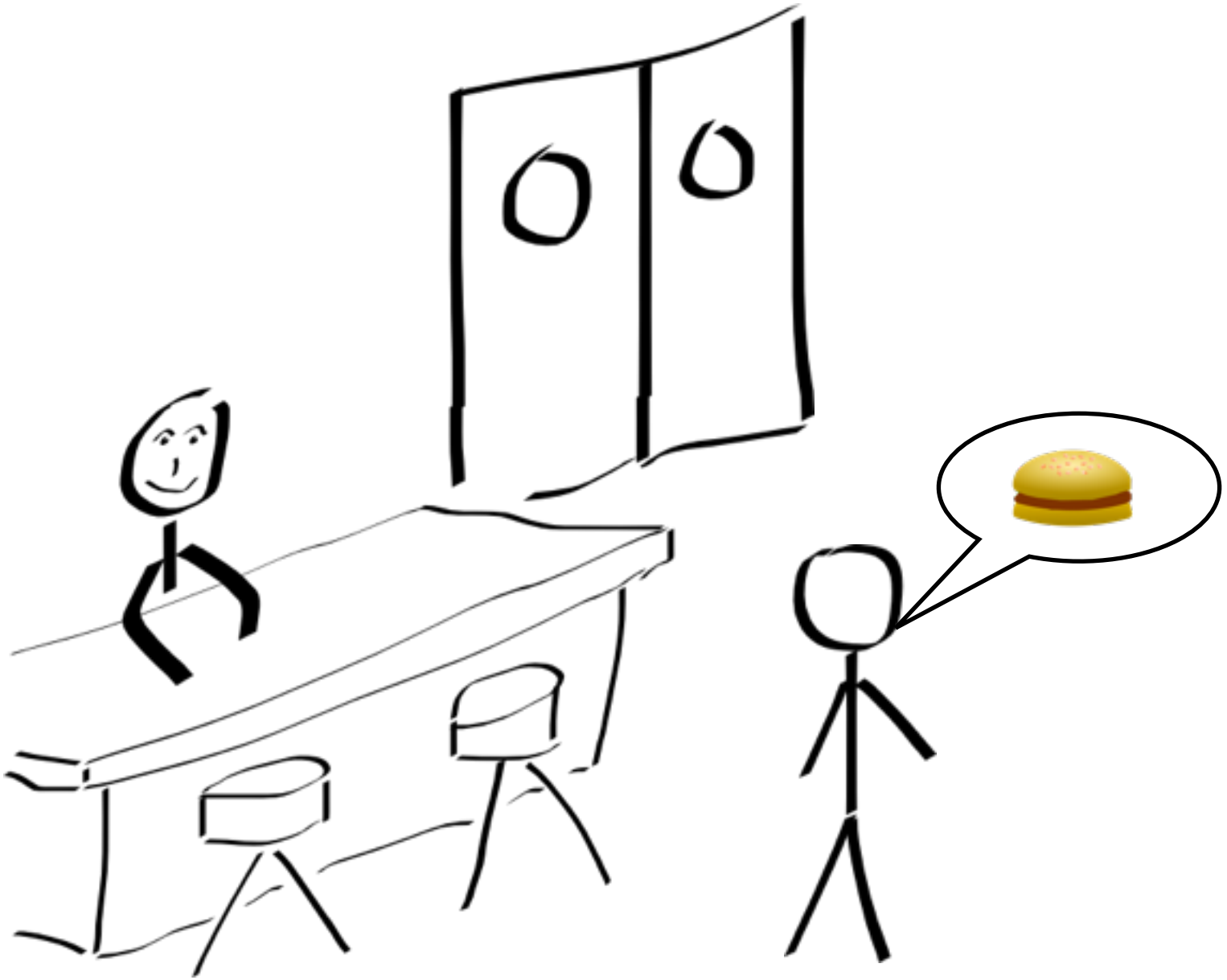


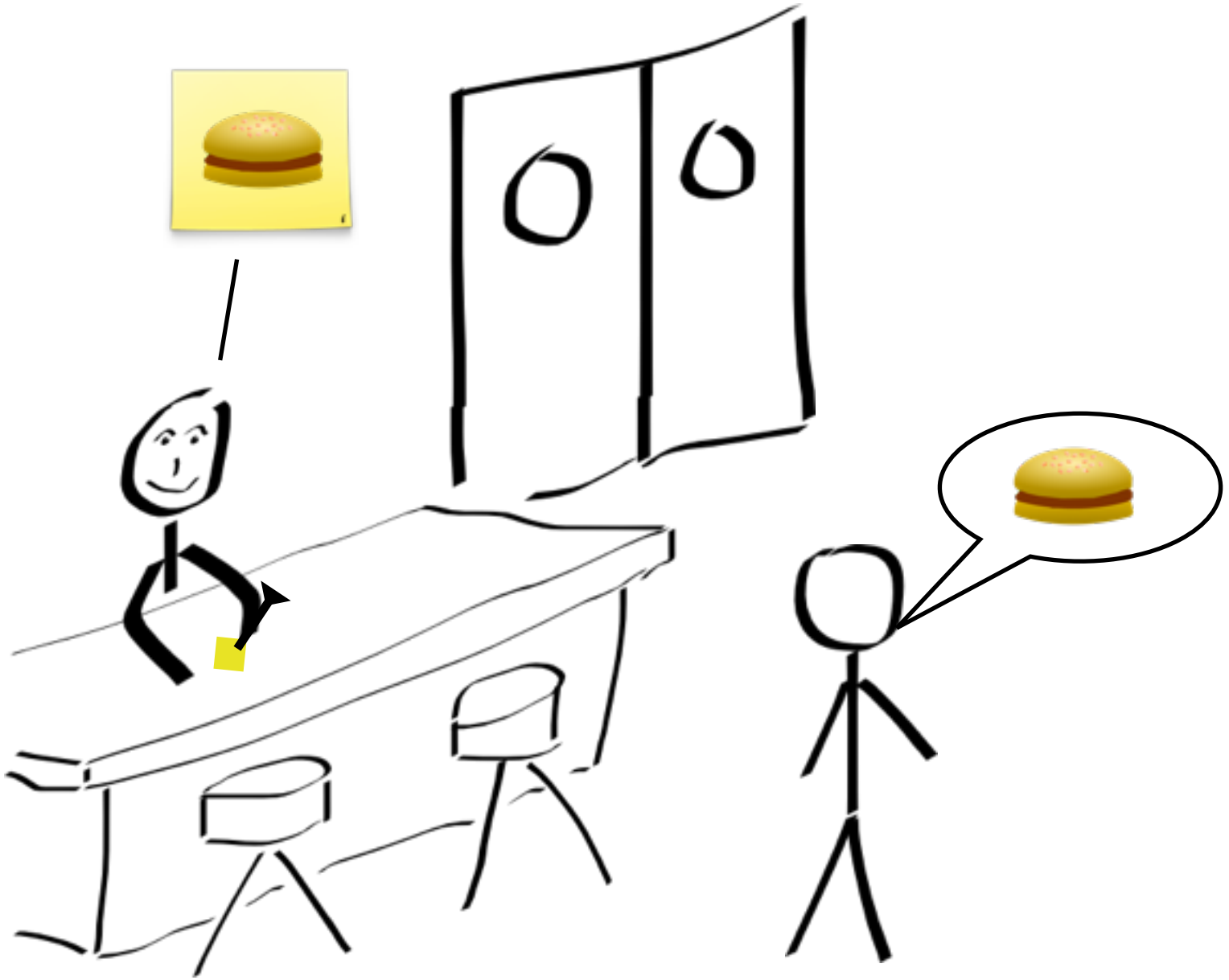

Ford
CRAN
COOKS

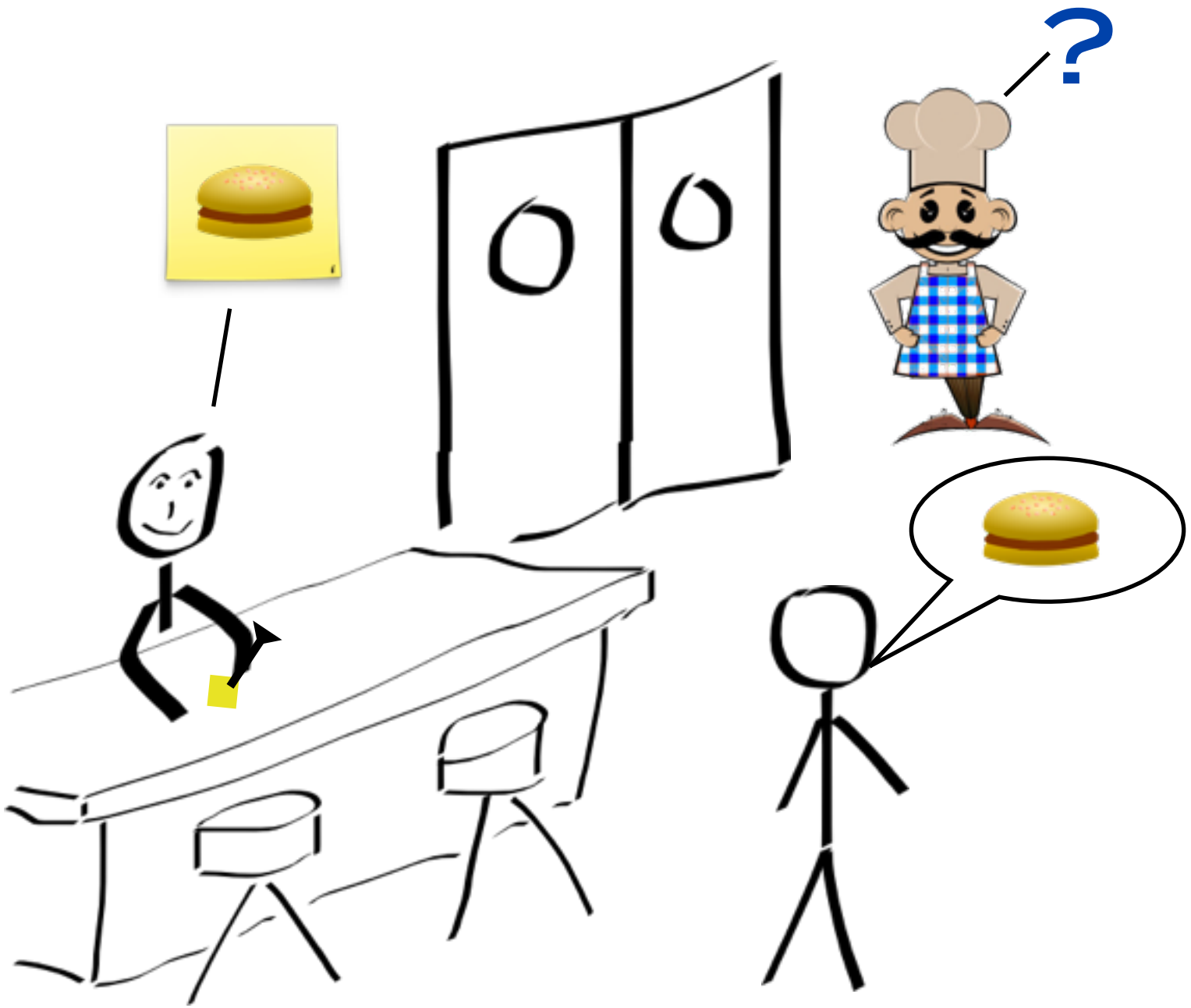
Whatever you want!

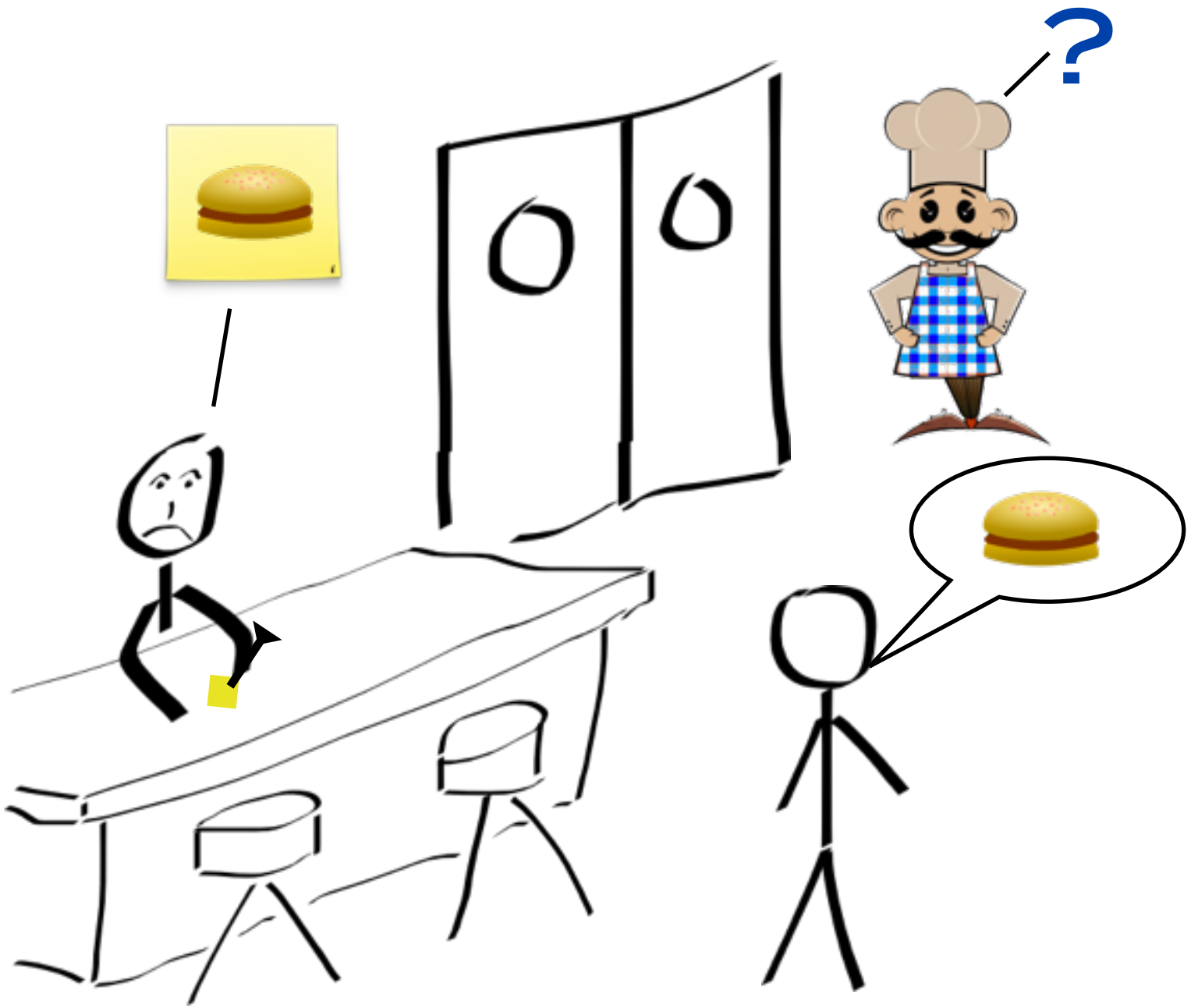

Ford
CRAN
COOKS

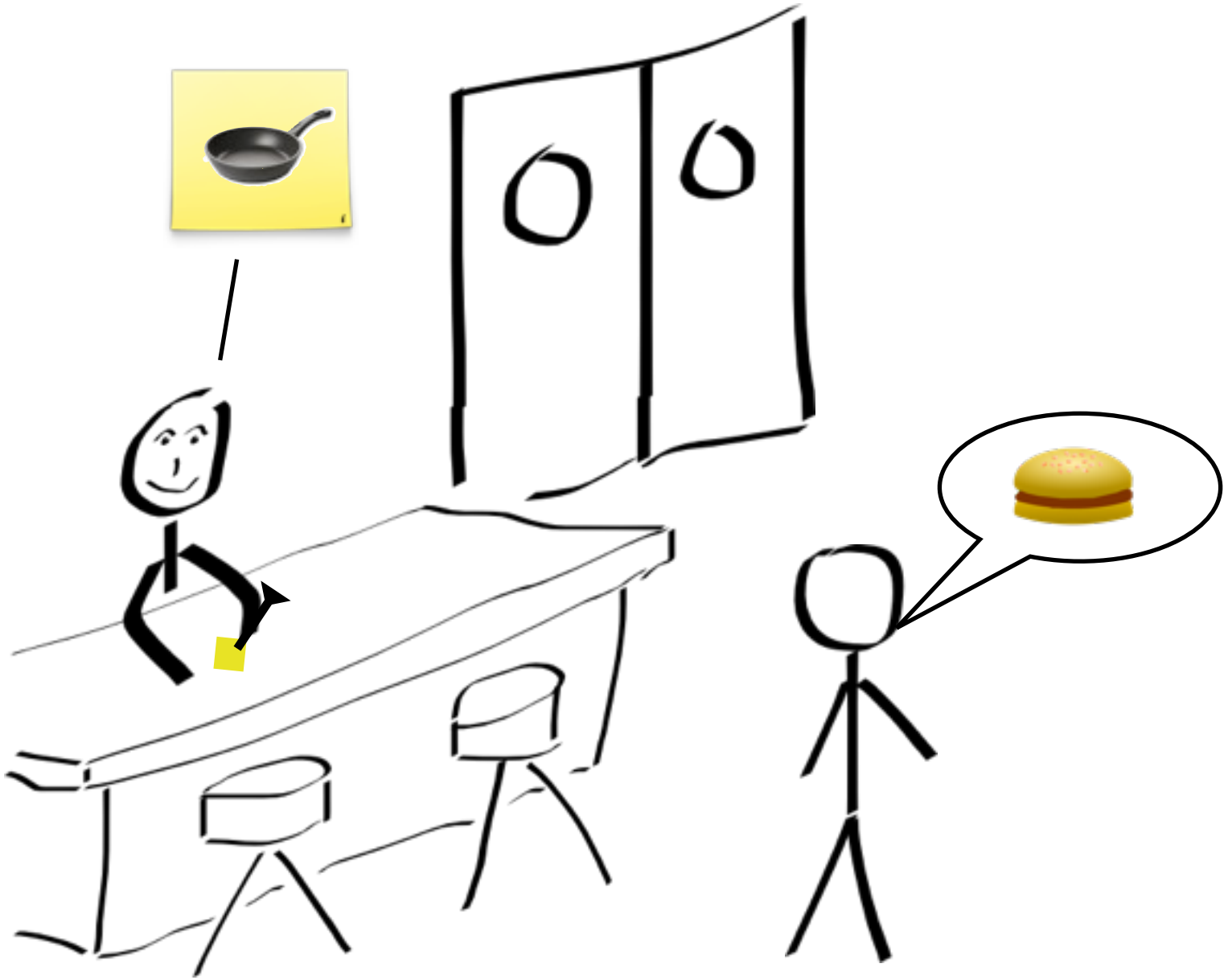


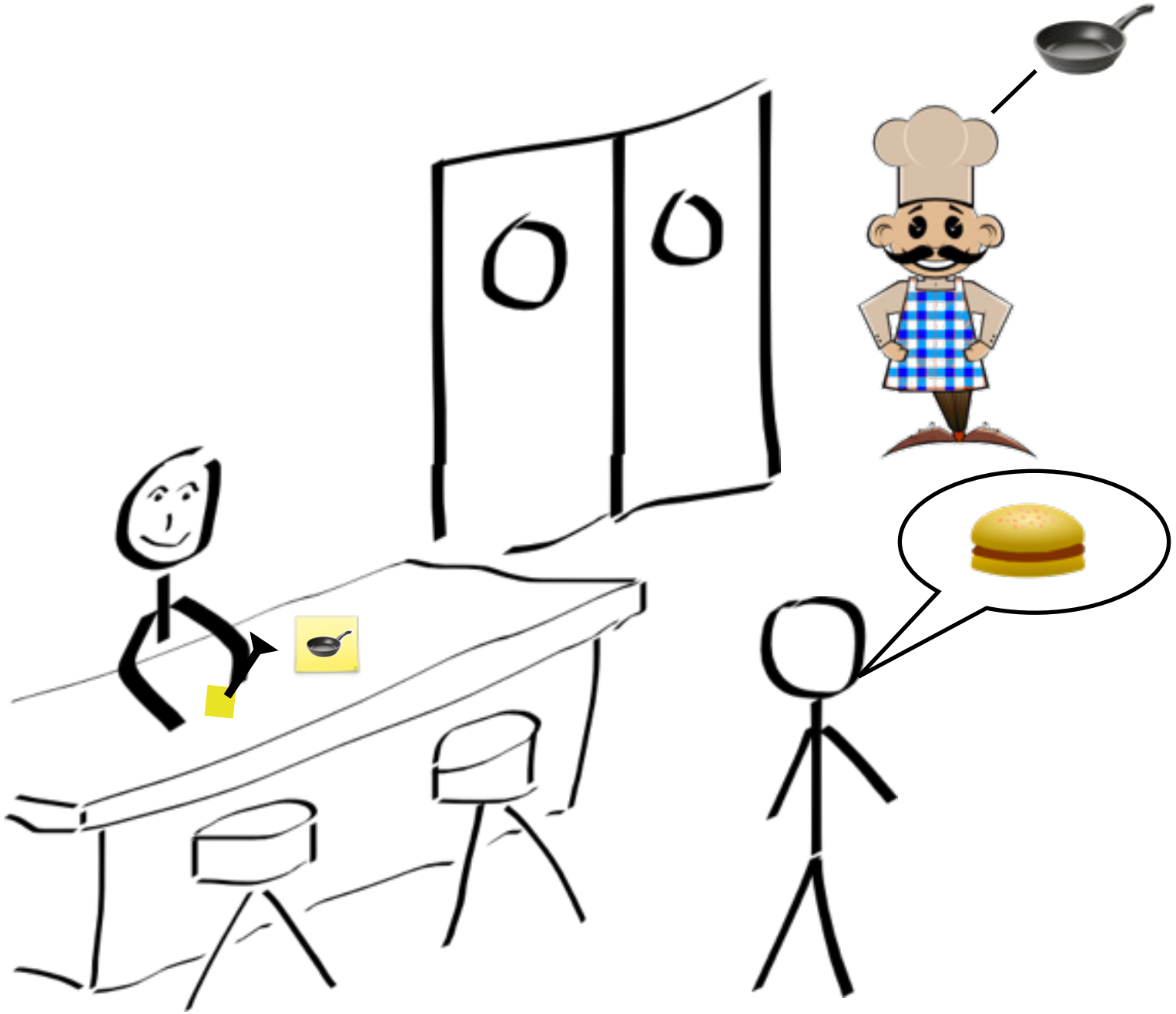


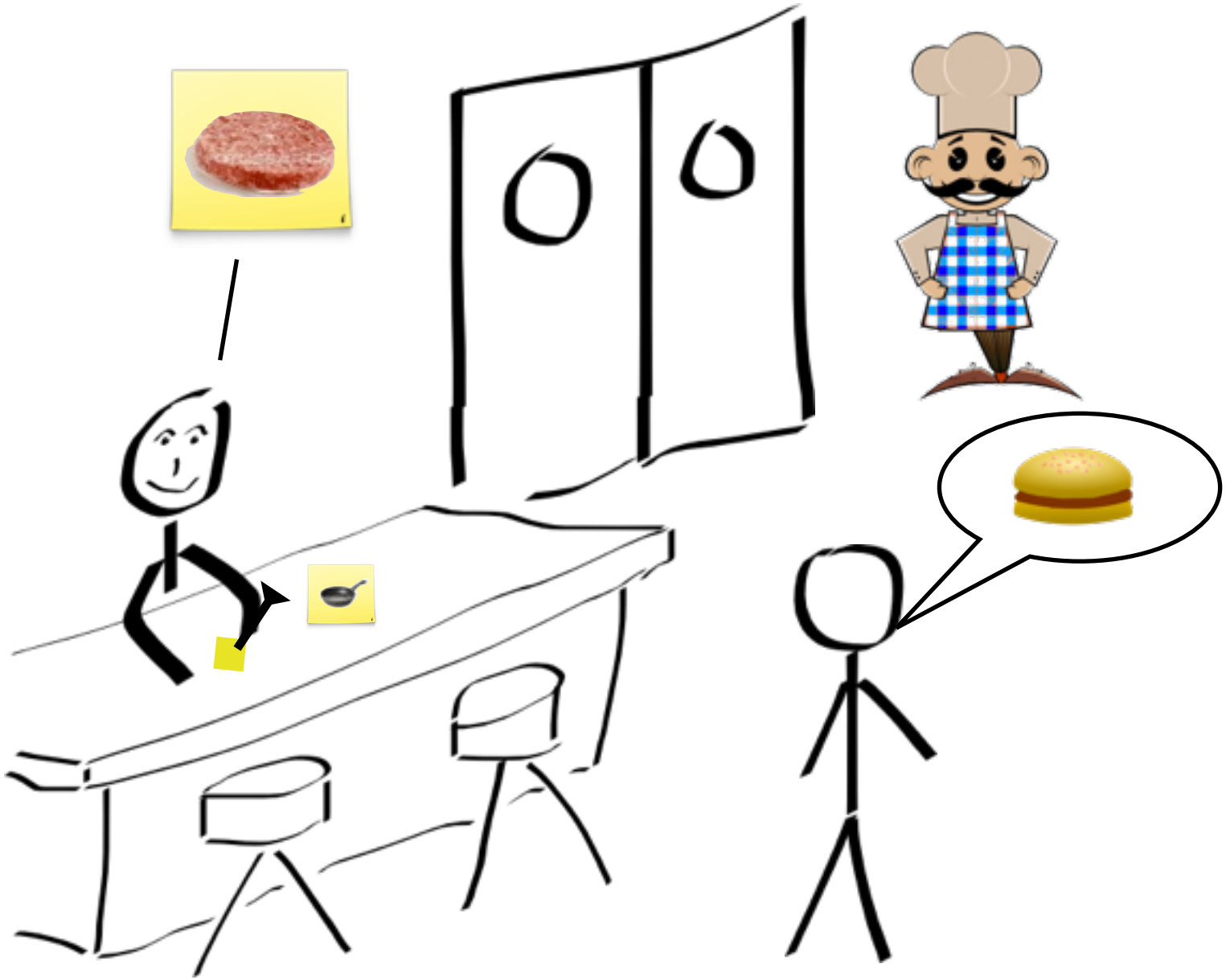


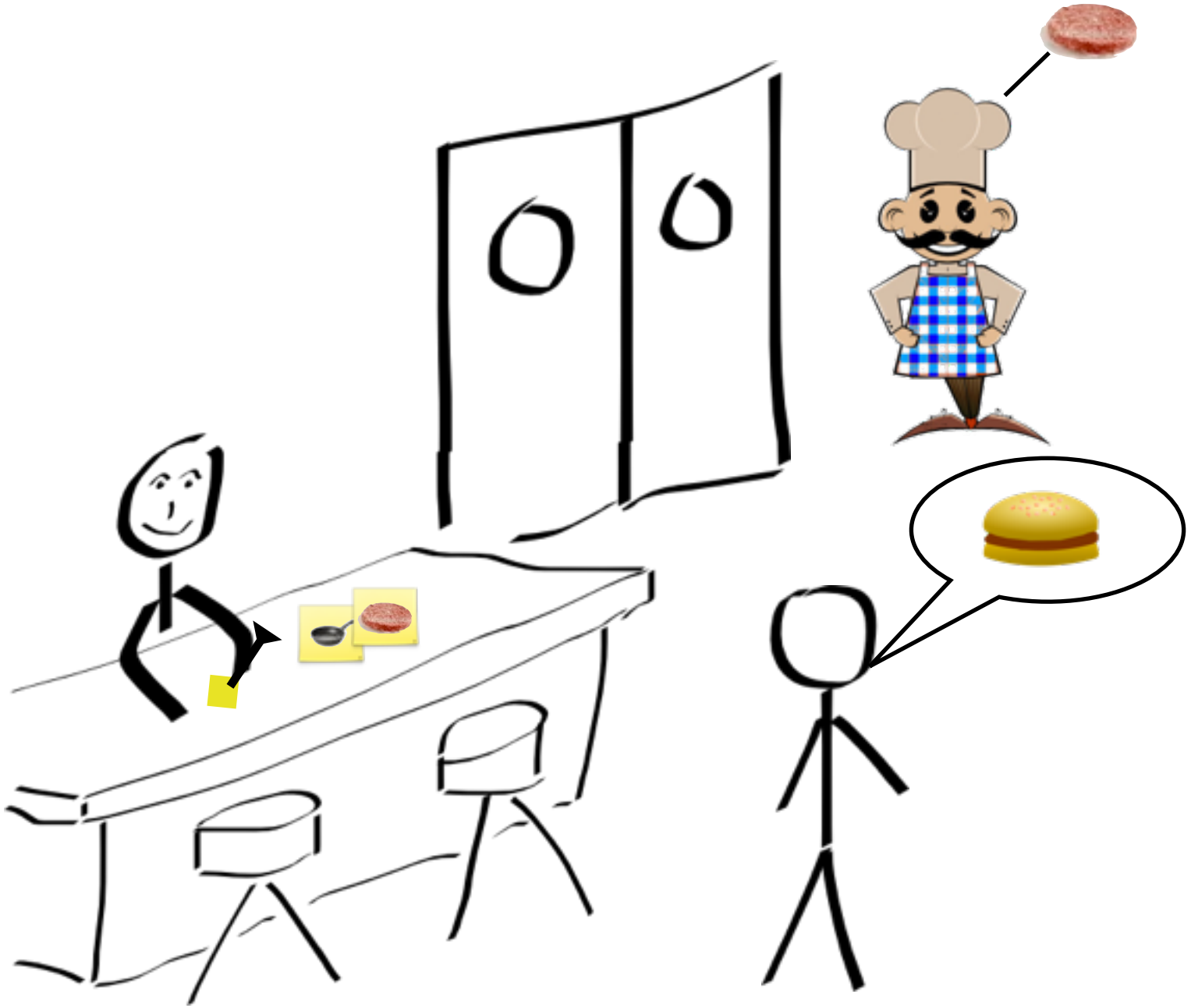


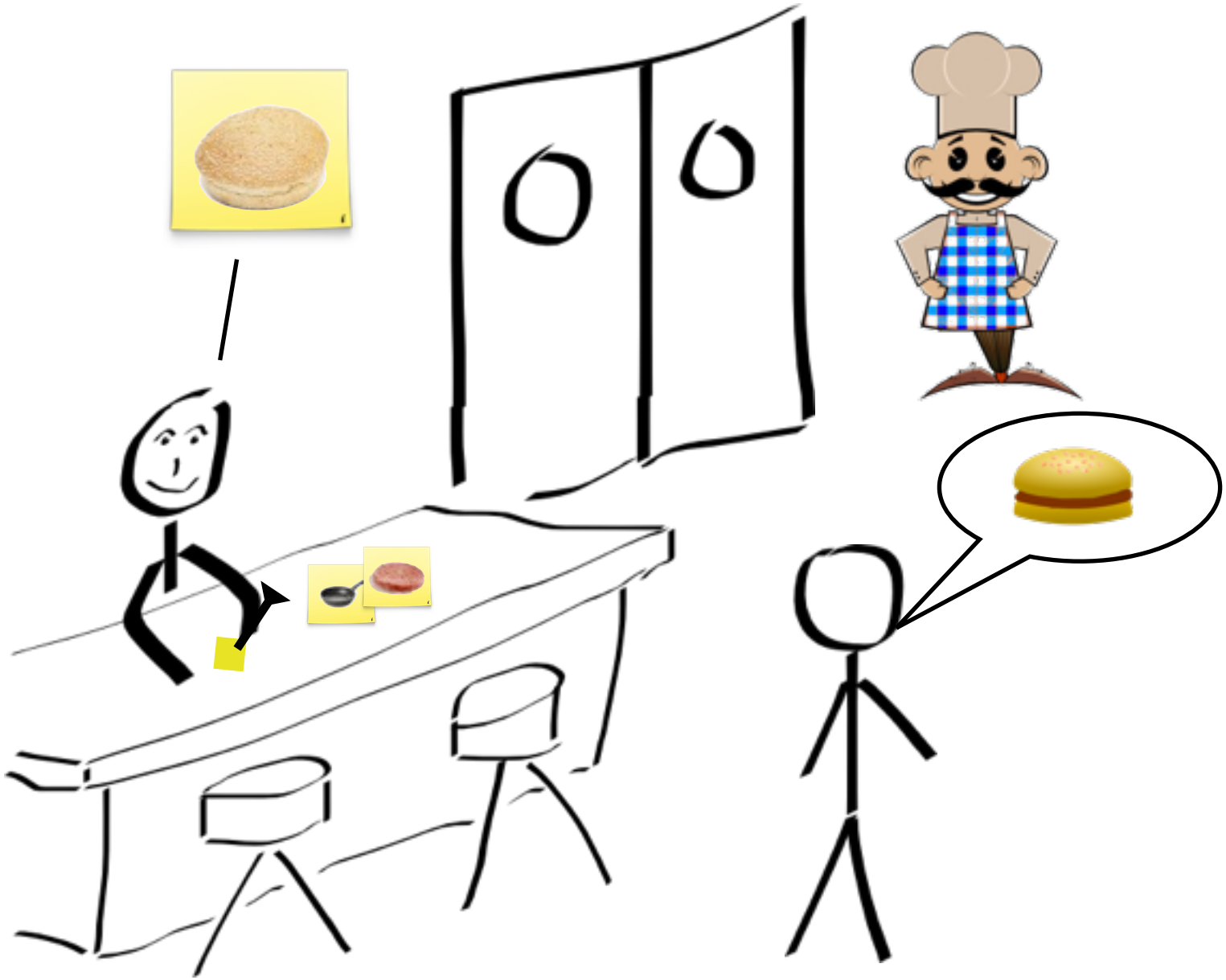


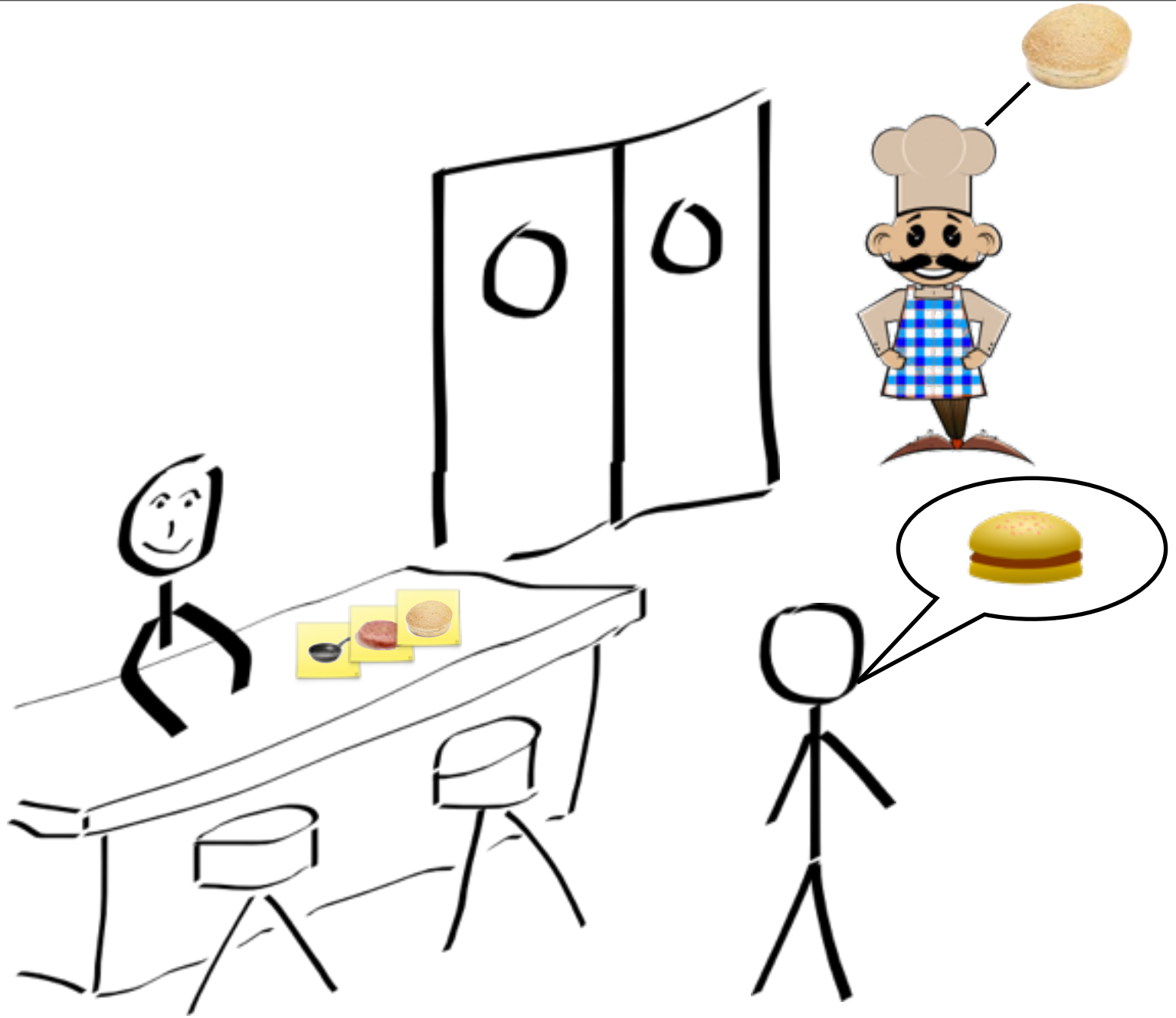


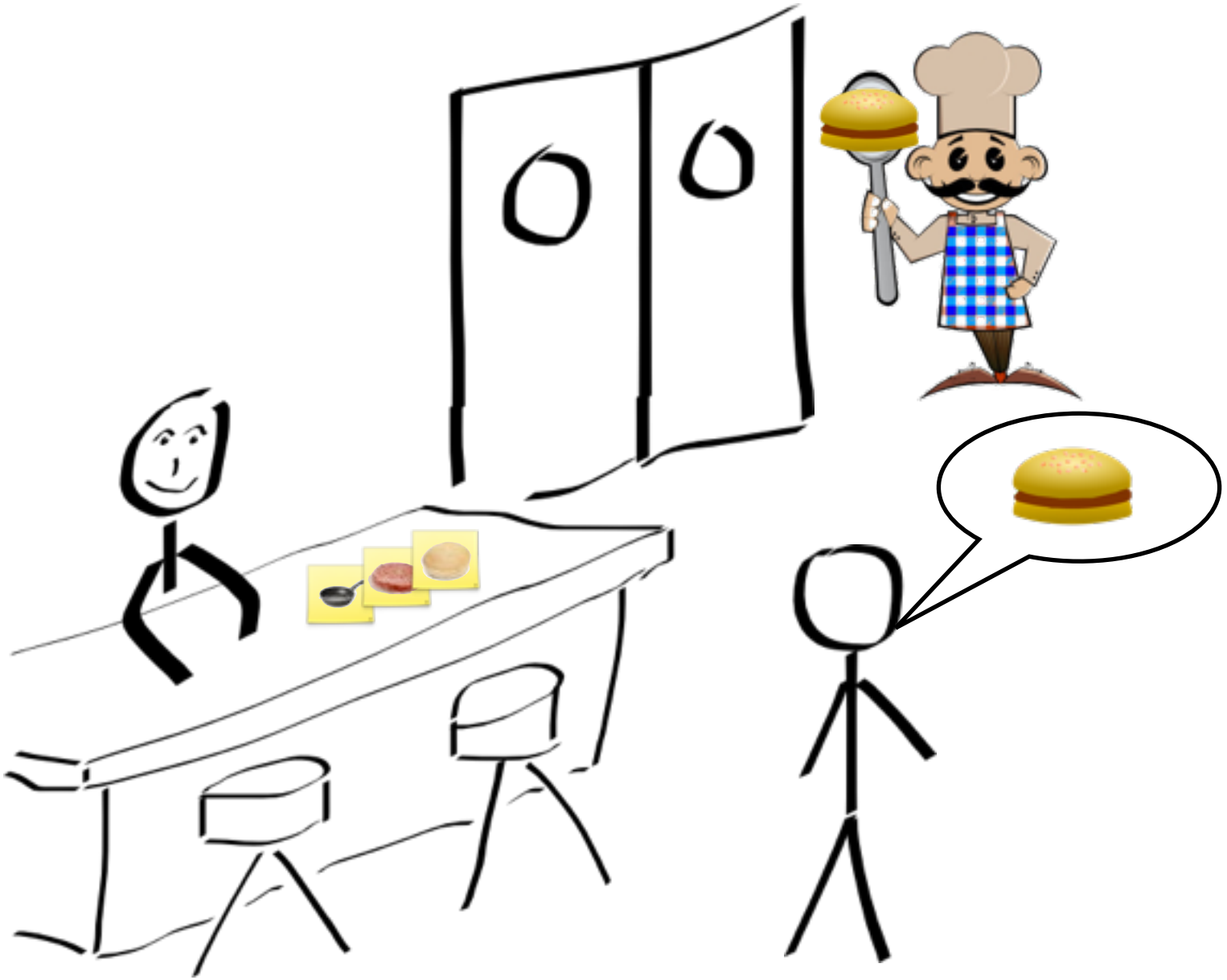


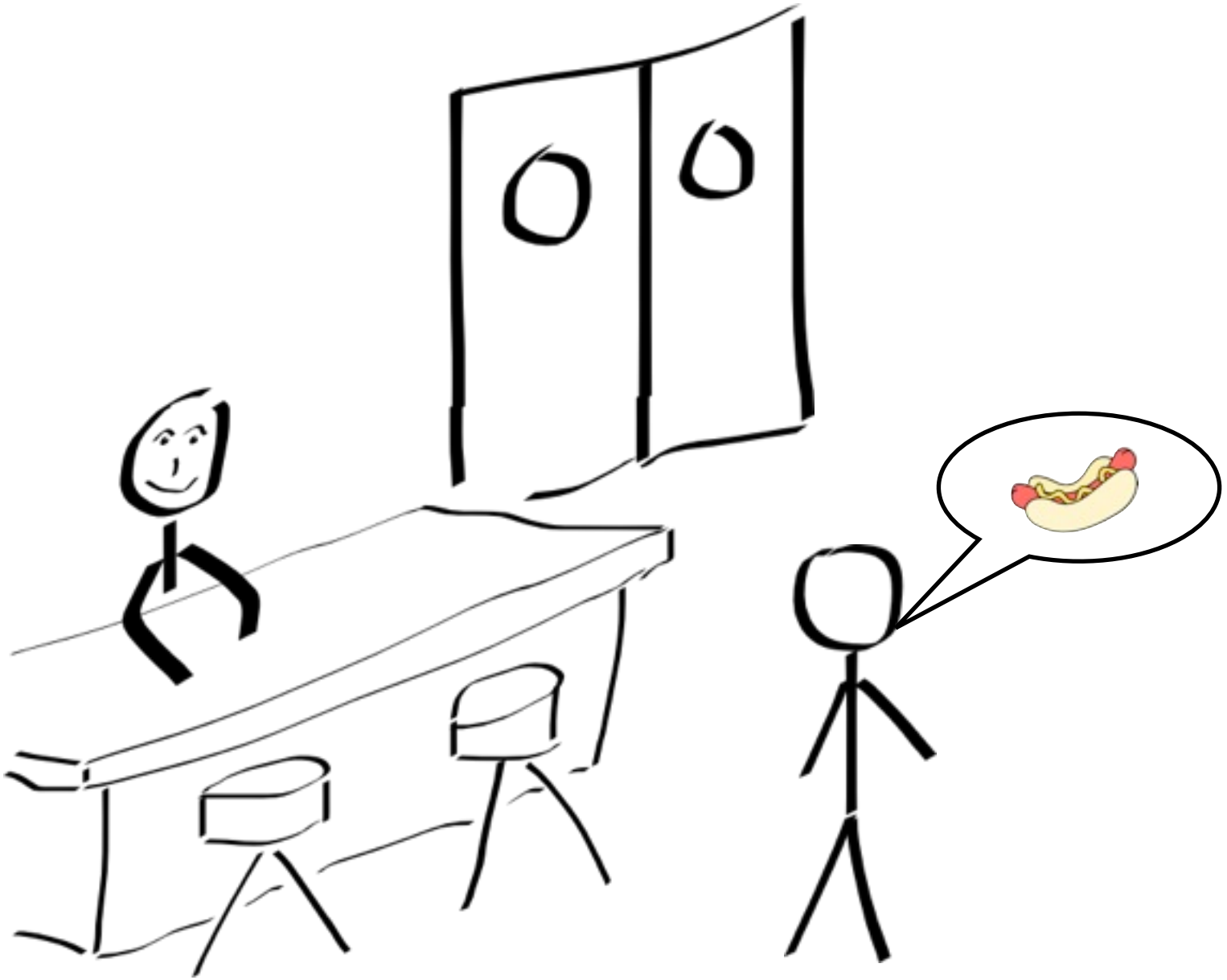






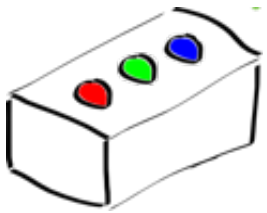


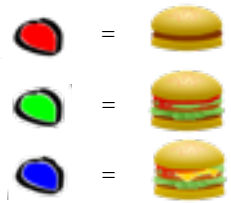


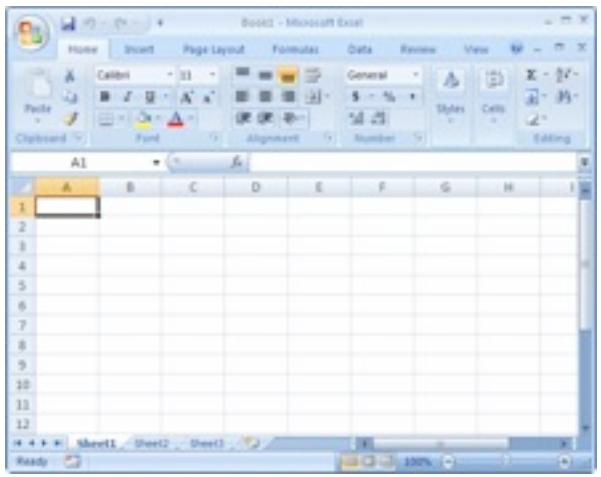



asd
asd

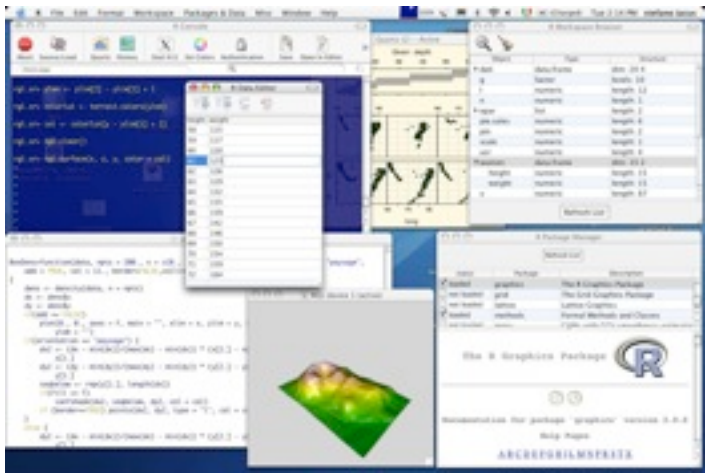

asd



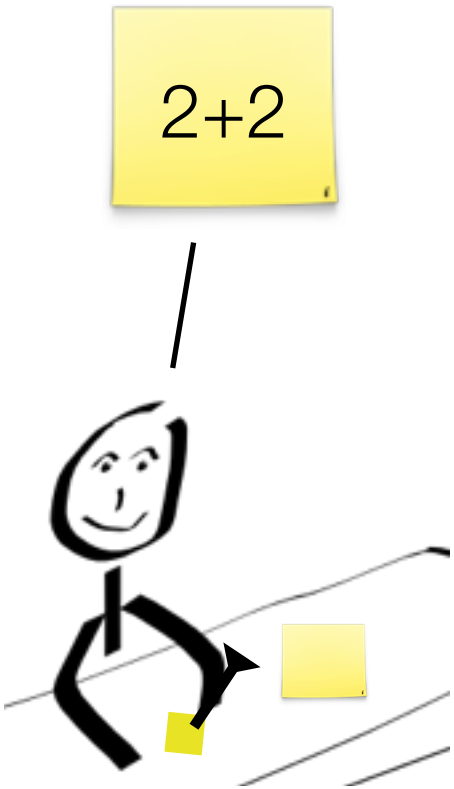


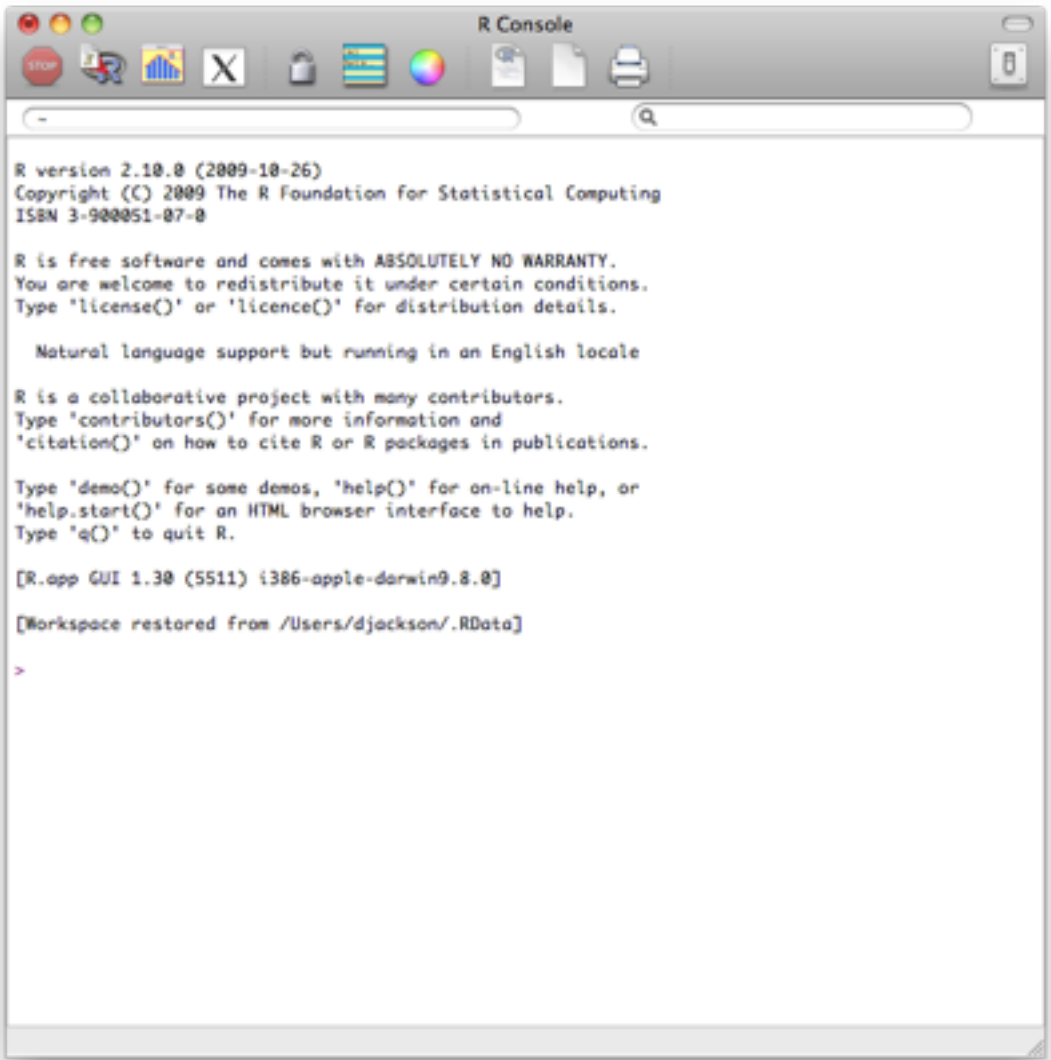


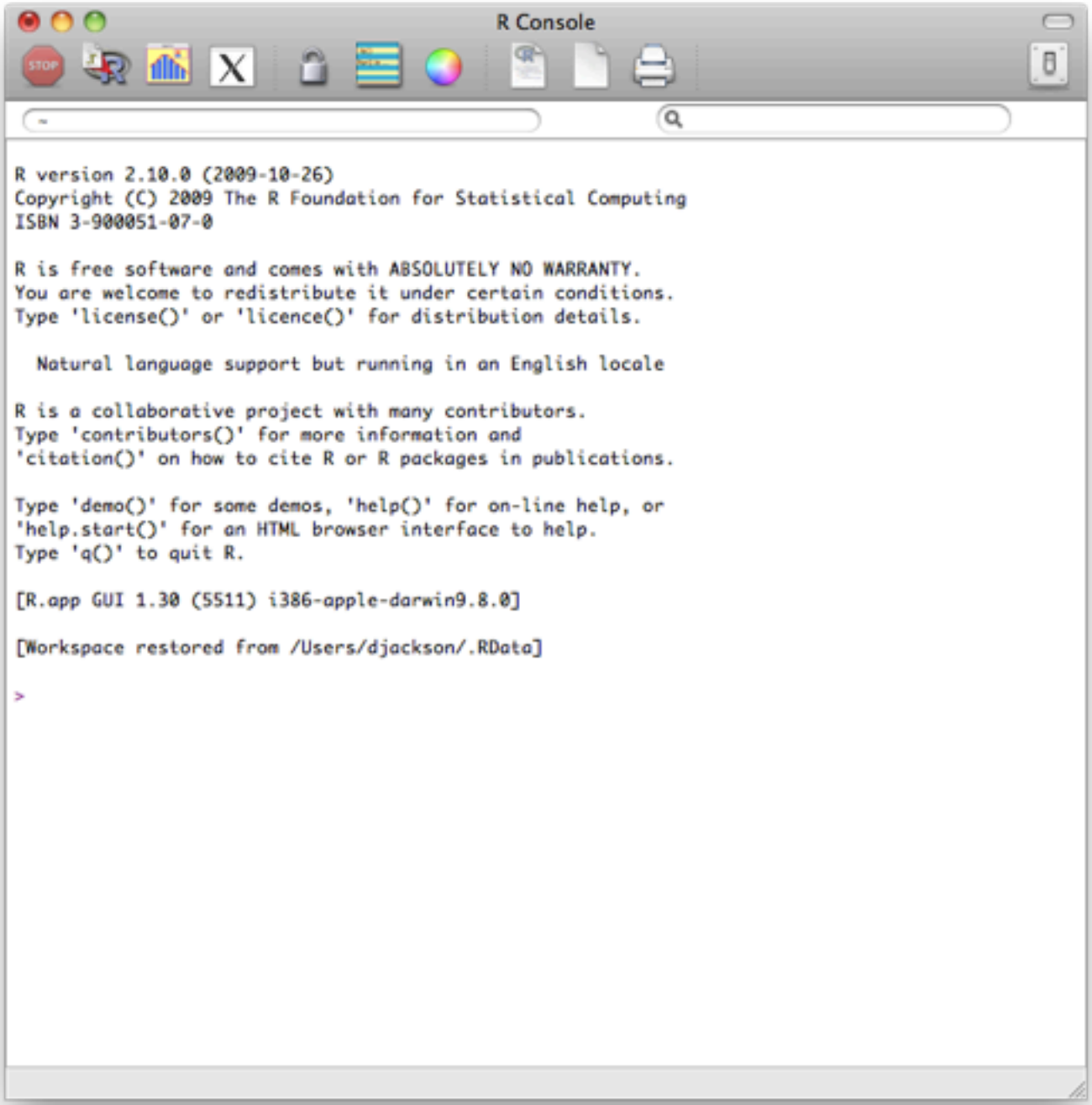
Whatever you want!

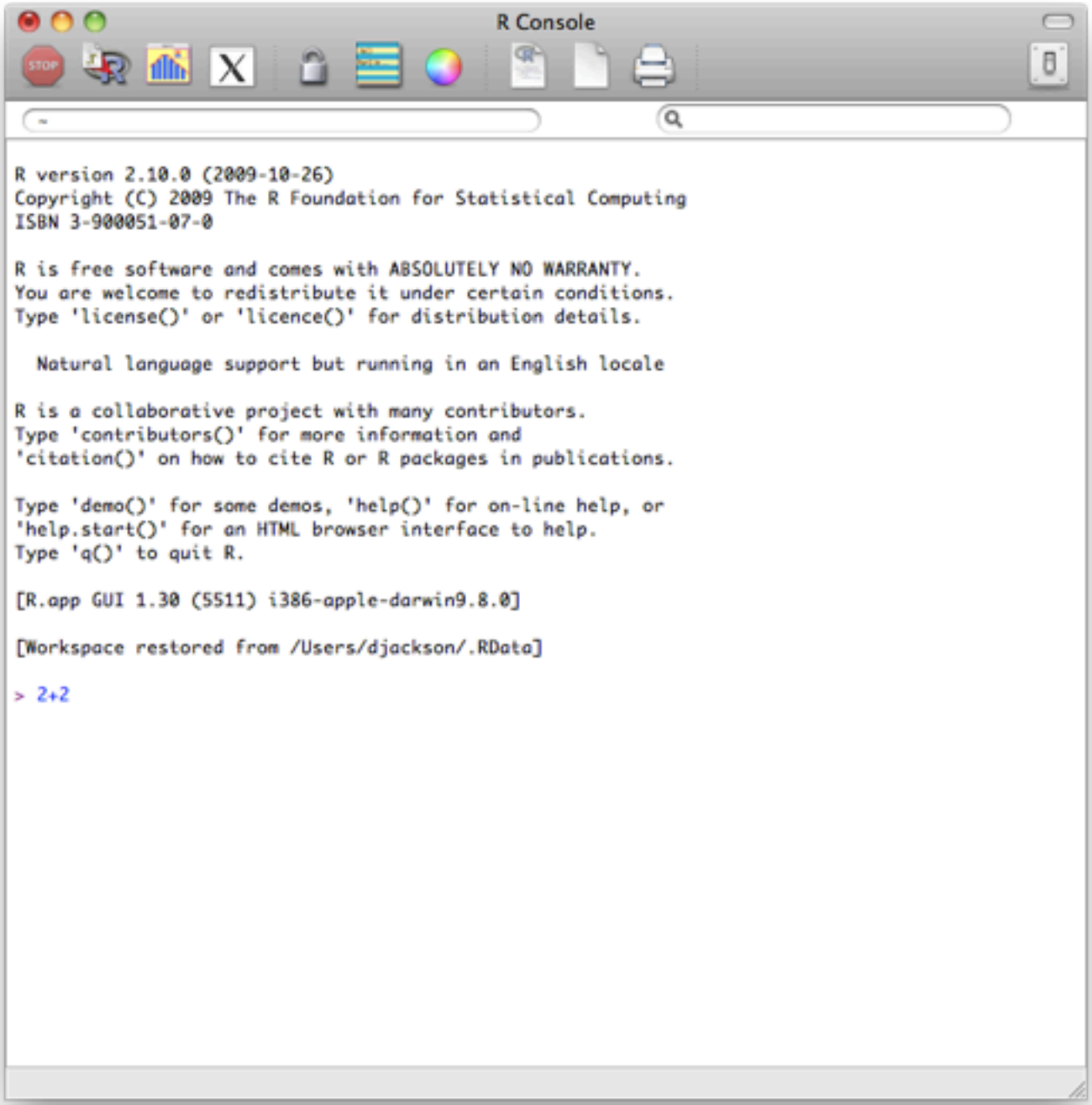


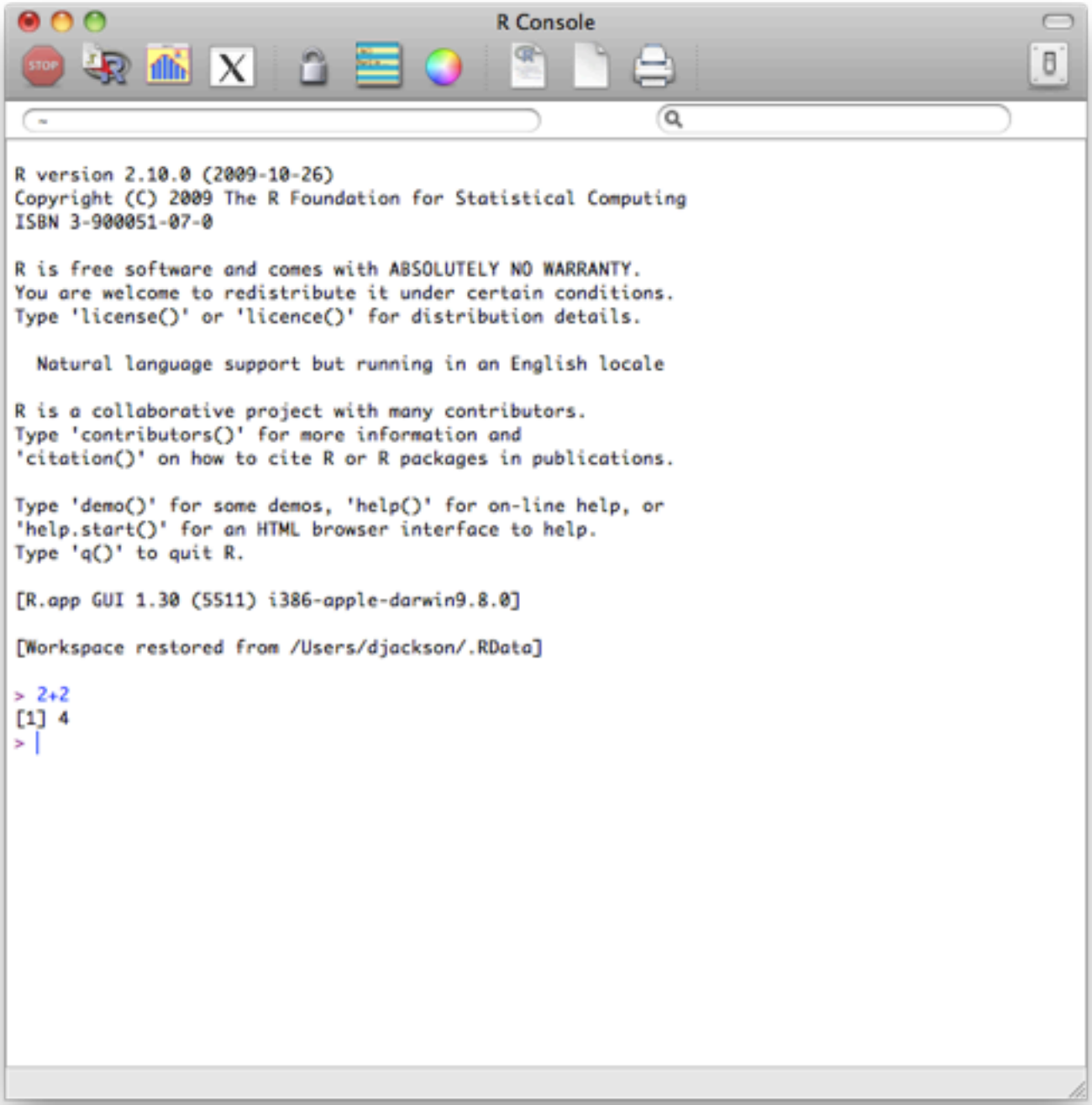
R Console


2+2


R Console
R version 2.10.0 (2009-10-26)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[R.app GUI 1.30 (5511) i386-apple-darwin9.8.0]
[Workspace restored from /Users/djackson/.RData]
>


R Console
R version 2.10.0 (2009-10-26)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[R.app GUI 1.30 (5511) i386-apple-darwin9.8.0]
[Workspace restored from /Users/djackson/.RData]
>


R Console
R version 2.10.0 (2009-10-26)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[R.app GUI 1.30 (5511) i386-apple-darwin9.8.0]
[Workspace restored from /Users/djackson/.RData]
> 2+2


R Console
R version 2.10.0 (2009-10-26)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[R.app GUI 1.30 (5511) i386-apple-darwin9.8.0]
[Workspace restored from /Users/djackson/.RData]
> 2+2
[1] 4
>

R Console

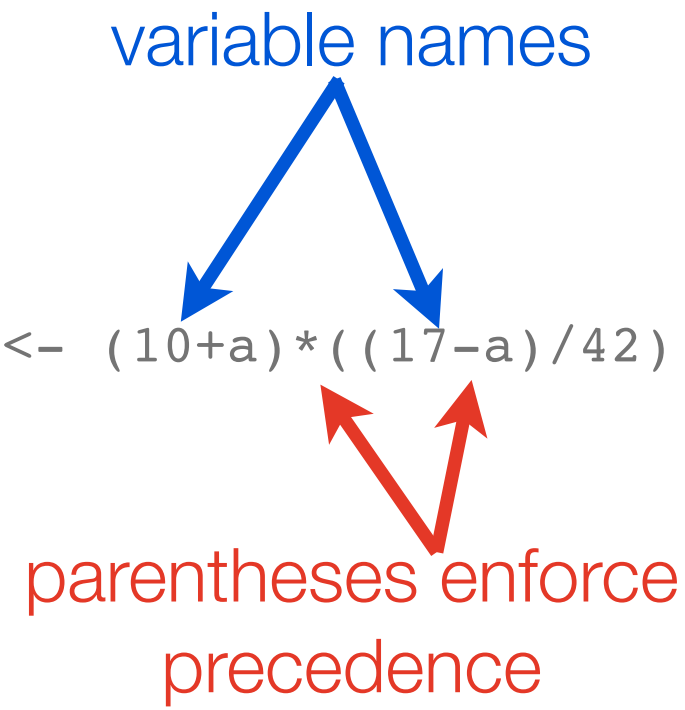
- command history
  - up/down arrows
  - history window

R Console

- assigning values
  - e.g., `a <- 1.375`

R Console

- other examples
  - subtraction: `a <- 62-20`
  - multiplication: `a <- 21*2`
  - division: `a <- 294/7`
  - complicated equations: `b <- (10+a)*((17-a)/42)`



R Console

- displaying values
  - e.g., a

R Console

- environment: what variables have I created?
  - `ls()`

R Console

- environment: what variables have I created?
  - `ls()`
- environment: how do I get rid of a variable?
  - e.g., `rm(a)`

R Console

- environment: what variables have I created?
  - `ls()`
- environment: how do I get rid of a variable?
  - e.g., `rm(a)`
- environment: how do I get rid of all my variables?
  - `rm(list=ls())`

R Console

- calling functions
  - e.g., `b <- round(a)`

R Console

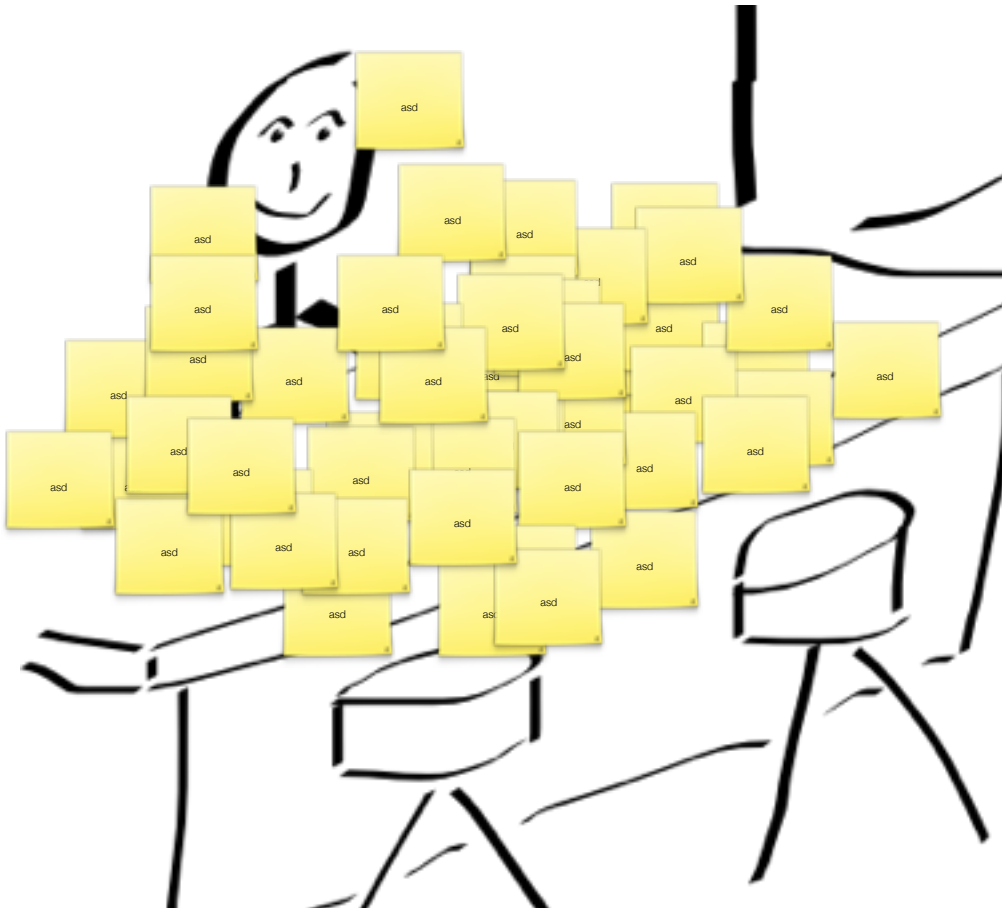
- getting help on a specific command or function
  - e.g., `help(round)`
  - or, equivalently, `?round`

R Console

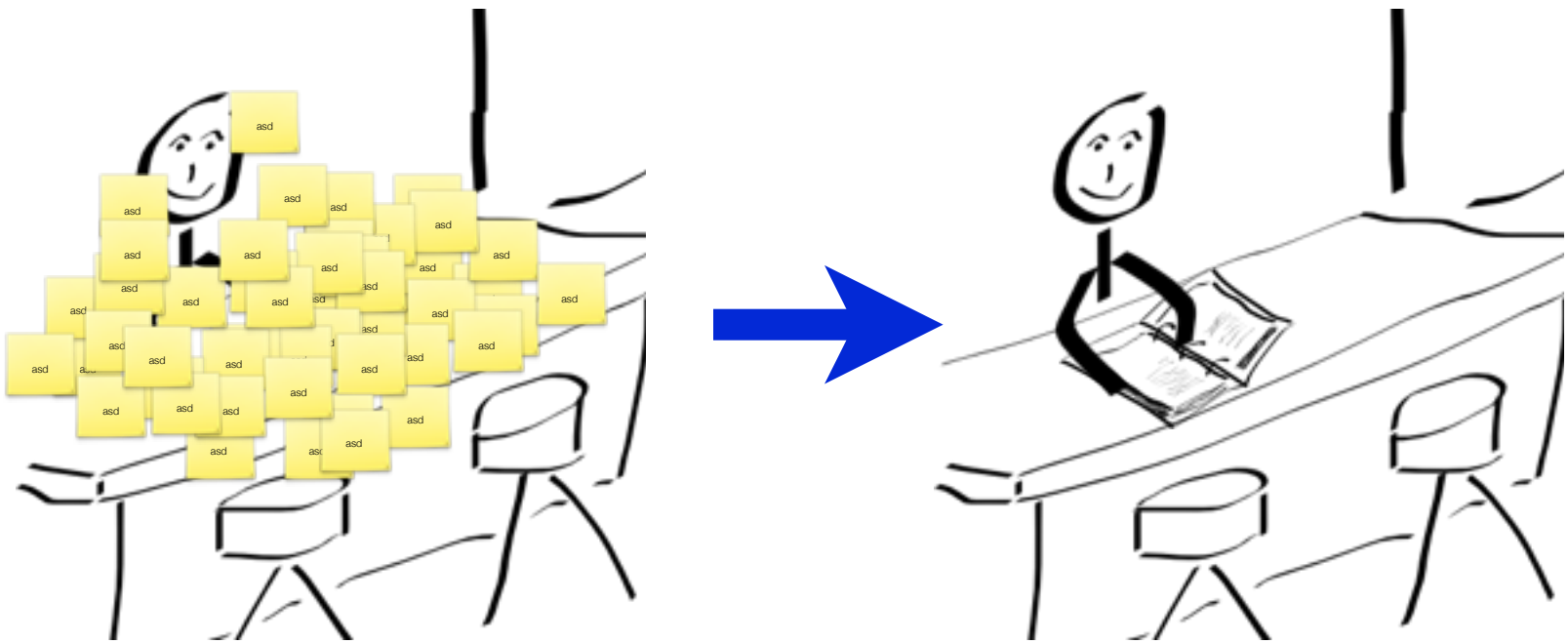
- getting general help using the R help browser

Exercise 1

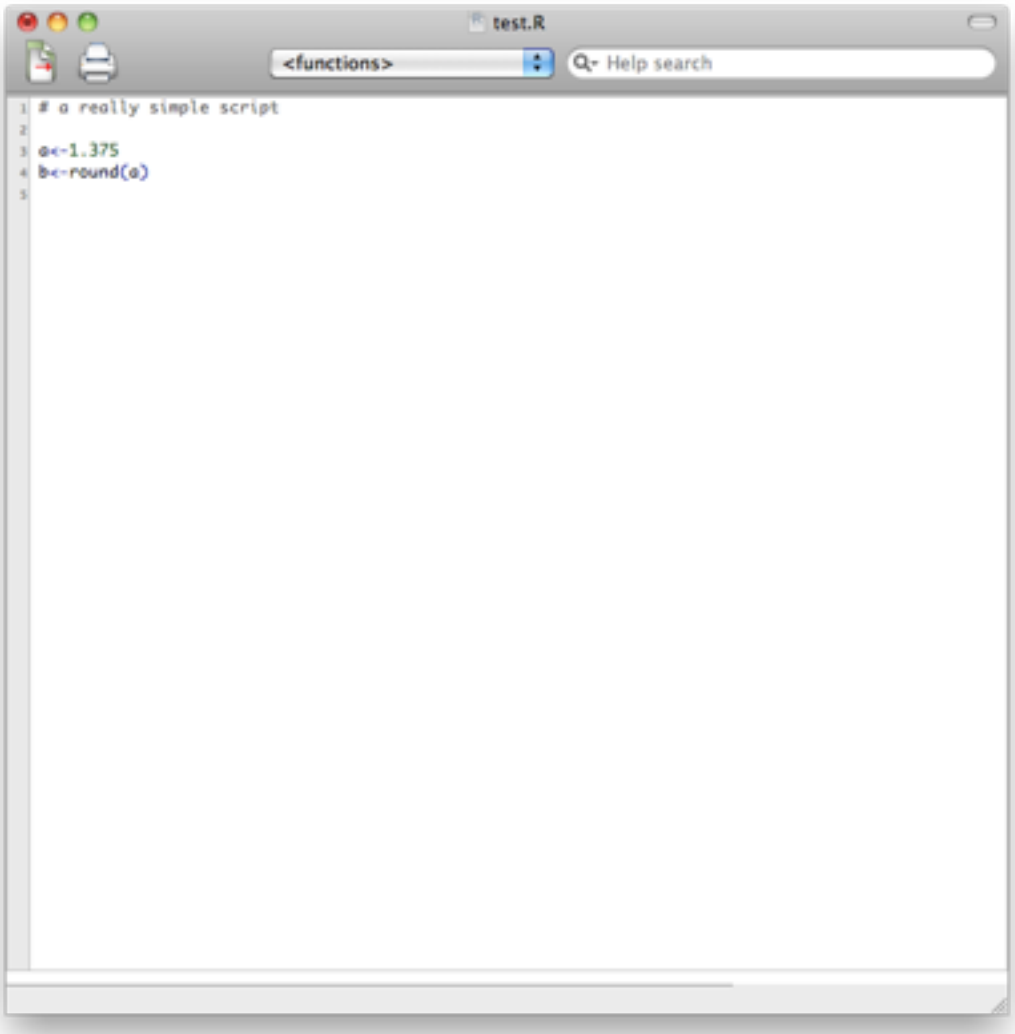
What about this problem?
And if you want to change a command?



R editor



R editor


test.R
<functions>
Help search
# a really simple script
a<-1.375
b<-round(a)

R editor

- Executing parts of your source code in the editor
  - ⌘-enter or Edit → Execute (Mac)
  - Ctrl-R or Edit → Run line or selection (Windows)

R editor

- Executing your entire source code file
  - `source('filename.R')`
  - ⌘E or Edit → Source Document (Mac)
  - Edit → Run all (Windows)

R editor

- Executing your entire source code file
  - `source('filename.R')`
  - ⌘E or Edit → Source Document (Mac)
  - Edit → Run all (Windows)
- Note: by default, results of commands aren't displayed
  - display all results: `source('filename.R', echo=TRUE)`
  - `round(a)` won't be displayed
  - `print(round(a))` will be displayed

R Console

- working directory
  - getting and setting from the command line
    - `getwd()`
    - `setwd('path_to_directory')`
  - getting and setting using menus

Exercise 2

Dealing with data

Vectors

Matrices

Data Frames

Lists (not covered here)

R has number of data structures, the simplest of which is a vector

```
> v <- c(1,2,3,4,5)
```

R has number of data structures, the simplest of which is a vector

```
> v <- c(1,2,3,4,5)
```

Whenever you give R a list you need the **c** command

R has number of data structures, the simplest of which is a vector

```
> v <- c(1,2,3,4,5)
> x <- 3
> w1 <- c(x, 2*x, x+1, x^3, -2)
> w1
[1] 3 6 4 27 -2
```

Values can be defined by previously stored variables

R has number of data structures, the simplest of which is a vector

```
> v <- c(1,2,3,4,5)
> x <- 3
> w1 <- c(x, 2*x, x+1, x^3, -2)
> w1
[1] 3 6 4 27 -2
> x <- 5
> w1
[1] 3 6 4 27 -2
```

Variables do not update, like an excel cell.

R has number of data structures, the simplest of which is a vector

```
> v <- c(1,2,3,4,5)
> x <- 3
> w1 <- c(x, 2*x, x+1, x^3, -2)
> w1
[1] 3 6 4 27 -2
> x <- 5
> w1
[1] 3 6 4 27 -2
> w2 <- c(w1, 5, x, v)
> w2
[1] 3 6 4 27 -2 5 3 1 2 3 4 5
```

c can combine vectors

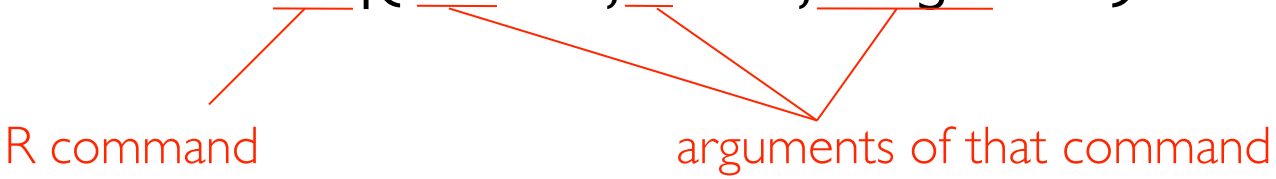
Regular sequences

```
> w3 <- seq(from=0,to=20,length=11)
```

Creating regular sequences with the R command **seq**

Regular sequences

```
> w3 <- seq(from=0,to=20,length=11)
```



Regular sequences

```
> w3 <- seq(from=0,to=20,length=11)
> w3
[1] 0 2 4 6 8 10 12 14 16 18 20
```

Regular sequences

```
> w3 <- seq(from=0,to=20,length=11)
> w3
[1] 0 2 4 6 8 10 12 14 16 18 20
> w4 <- seq(from=0,to=20,by=4)
[1] 0 4 8 12 16 20
```

An alternative way to use the `seq` command, or you can use `from`, `length` and `by`. Generally you will not specify every argument when you call an R command. Use `?seq` to see the full list of arguments.

Vector operations

```
   > v
[1] 1 2 3 4 5
   > w1
[1] 3 6 4 27 -2
```

Vector operations

```
   > v
[1] 1 2 3 4 5
   > w1
[1] 3 6 4 27 -2
   > w5 <- 2*v - w1 + 5
   > w5
[1] 4 3 7 -14 17
```

Vector operations

```
   > v
[1] 1 2 3 4 5
   > w1
[1] 3 6 4 27 -2
   > w5 <- 2*v - w1 + 5
   > w5
[1] 4 3 7 -14 17
```

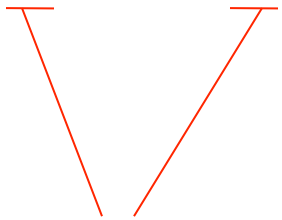
Now you are ready for Exercise 3.

Fuel economy with vectors.

```
> fe_city_vec <- seq(from=20,to=40,by=0.25)
> fe_high_vec <- fe_city_vec*1.25
> 1/((0.55/fe_city_vec)+ (0.45/fe_high_vec))
```

Vectors can take also character values in addition to numeric values

```
> s <- c("site a", "site b", "site b", "site a",
        "site a")
```



To assign character values you need to use quotes (single or double), otherwise R thinks you are referencing a previously stored variable.

Vectors can take also character values in addition to numeric values

```
> s <- c("site a", "site b", "site b", "site a",
        "site a")
```

Each vector has a **mode**, that is the type of data it contains.

```
> mode(s)
[1] "character"
> mode(v)
[1] "numeric"
```

Vectors can take also character values in addition to numeric values

```
> s <- c("site a", "site b", "site b", "site a",
        "site a")
```

Each vector has a **mode**, that is the type of data it contains.

```
> mode(s)
[1] "character"
> mode(v)
[1] "numeric"
```

A vector can only have a single mode.

```
> t <- c("a",1)
> t
[1] "a" "1"
> mode(t)
[1] "character"
```

The second element of `t` is the character `"1"` not the number `1`. Since the first entry was a character R forced all entries to be a character since vectors can have only a single mode

You can 'force' vectors into other modes.

```
> t1 <- as.numeric(t)
Warning message:
NAs introduced by coercion
 > t1
[1] NA 1
```

The character "1" was changed to the number 1.

Since "a" is not a number, R changed it to **NA**, the only non-number allowed in numeric vectors.

You can 'force' vectors into other modes.

```
  > t1 <- as.numeric(t)
Warning message:
NAs introduced by coercion
 > t1
[1] NA 1
 > t2 <- as.character(v)
 > t2
[1] "1" "2" "3" "4" "5"
```

Vector indexing

```
   > w2
[1] 3 6 4 27 -2 5 3 1 2 3 4 5
   > w2[6]
[1] 5
   > w2[6] <- -5
```

Use square brackets to reference or redefine a specific element of a vector

Vector indexing

```
   > w2
[1] 3 6 4 27 -2 5 3 1 2 3 4 5
   > w2[6]
[1] 5
   > w2[6] <- -5
   > w2[2:9]
[1] 6 4 27 -2 5 3 1 2
```

Brackets can also be used to get more than one value of a vector.
Colon means all integers between.

Vector indexing

```
   > w2
[1] 3 6 4 27 -2 5 3 1 2 3 4 5
   > w2[6]
[1] 5
   > w2[6] <- -5
   > w2[2:9]
[1] 6 4 27 -2 -5 3 1 2
   > w6 <- w2[5:12]
   > w6
[1] -5 3 1 2 3 4 5
```

Can be used to create a new vector that is a subset of an existing one.

Indexing with logical operators

```
  > s
[1] “site a” “site b” “site b” “site a” “site a”
  > w1
[1] 3 6 4 27 -2
  > w5
[1] 4 3 7 -14 17
```

Indexing with logical operators

```
  > s
[1] “site a” “site b” “site b” “site a” “site a”
  > w1
[1] 3 6 4 27 -2
  > w5
[1] 4 3 7 -14 17
  > w1[s == “site a”]
[1] 3 27 -2
```

This command returns values of **w1** for which the corresponding value of **s** is equal to “site a”. In logical expressions like this ‘is equal to’ is denoted **==**.

Indexing with logical operators

```
  > s
[1] “site a” “site b” “site b” “site a” “site a”
  > w1
[1] 3 6 4 27 -2
  > w5
[1] 4 3 7 -14 17
  > w1[s == “site a”]
[1] 3 27 -2
  > w5[w1 > 3.5]
[1] 3 7 -14
```

Similarly we can find values of **w5** such that corresponding values of **w1** are greater than 3.5.

Indexing with logical operators

```
  > s
[1] “site a” “site b” “site b” “site a” “site a”
  > w1
[1] 3 6 4 27 -2
  > w5
[1] 4 3 7 -14 17
  > w1[s == “site a”]
[1] 3 27 -2
  > w5[w1 > 3.5]
[1] 3 7 -14
  > w1[w1 > 5.5 & s == “site b”]
[1] 6
```

Logical conditions can be combined with and (&).

List of logical operators used for indexing

R command	meaning
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
==	equal to
!	not
&	and
|	or

The `length` command

```
  > w2
[1] 3 6 4 27 -2 5 3 1 2 3 4 5
  > length(w2)
[1] 12
  > length(w2[s == “site a”])
[1] 3
  > length(w2[w2 > 4.5]
[1] 4
```

Tells you how long a vector is. It is often used to ‘count’ how many of something fulfill a certain condition.

The `length` command

```
  > w2
[1] 3 6 4 27 -2 5 3 1 2 3 4 5
  > length(w2)
[1] 12
  > length(s[s == “site a”])
[1] 3
  > length(w2[w2 > 4.5]
[1] 4
```

Now ready for Exercise 4. You will also need the commands `sum` and `max`, they work like `length`. Remember you can use `?sum` to get help.

How many PMs were from the state of Tasmania?

```
> length(names[state_rep==“Tasmania”])
[1] 1
```

Fraction of time Labor and Liberal parties in power

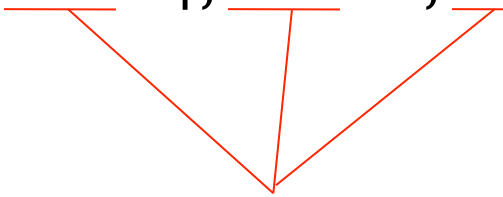
```
> sum(days[party == “Liberal”])/sum(days)
[1] 0.383
> sum(days[party == “Labor”])/sum(days)
[1] 0.371
```

Which prime minister served longest?

```
> pm_name[days == max(days)]
[1] Robert Menzies
> max(days)/365.25
[1] 16.10
```

Once you have vectors matrices are the logical next step

```
> q <- seq(from=1,to=16,length=16)
> M1 <- matrix(data=q,nrow=4,ncol=4)
```



You can turn an existing vector into a matrix with the **`matrix`** command. Note the three arguments.

Once you have vectors matrices are the logical next step

```
> q <- seq(from=1,to=16,length=16)
> M1 <- matrix(data=q,nrow=4,ncol=4)
> M1
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

Once you have vectors matrices are the logical next step

```
> q <- seq(from=1,to=16,length=16)
> M2 <- matrix(data=q,nrow=4,ncol=4,byrow=TRUE)
> M2
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
```

You can make the data fill in the other way with the `byrow` argument.

Another way to define a matrix

```
> M3 <- rbind(w1,w5,v)
   [,1] [,2] [,3] [,4] [,5]
w1    3    6    4   27   -2
w5    4    3    7  -14   17
v     1    2    3    4    5
```

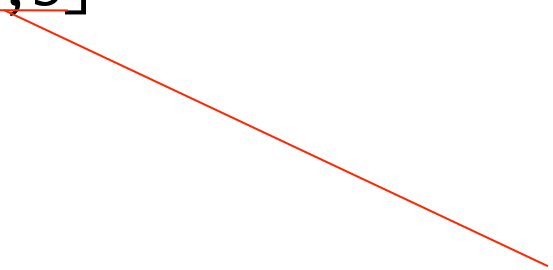
Create a matrix whose rows are existing vectors.

```
> M4 <- cbind(w1,w5,v)
      w1   w5   v
[1,]   3    4   1
[2,]   6    3   2
[3,]   4    7   3
[4,]  27  -14   4
[5,]  -2   17   5
```

Create a matrix whose columns are existing vectors.

Matrix indexing

```
> M1[2,3]
[1] 10
```



First specify the row and then the column, separated by a comma.

Matrix indexing

```
  > M1[2,3]
[1] 10
  > M1[1:2,2:4]
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
```

The colon functions in the same manner as with vectors

Matrix indexing

```
  > M1[2,3]
[1] 10
  > M1[1:2,2:4]
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
  > M1[2,]
[1] 2 6 10 14
  > M1[,2]
[1] 5 6 7 8
```

Leaving the space after or before the comma blank gives a whole row or column

Matrix indexing

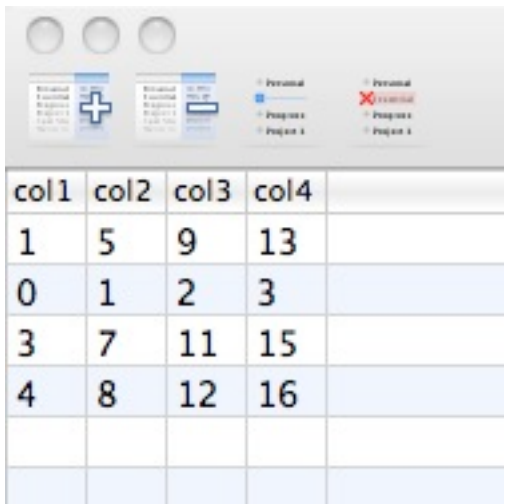
```
  > M1[2,3]
[1] 10
  > M1[1:2,2:4]
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
  > M1[2,]
[1] 2 6 10 14
  > M1[,2]
[1] 5 6 7 8
  > M1[2,] <- c(0,1,2,3)
```

These commands can be used to redefine sections of a matrix.

Matrices and vectors can be manipulated in a spreadsheet-like environment

```
> M6 <- edit(M1)
```

The command **`edit`** opens up the spreadsheet window, but in order for the changes you make in it to be saved you must assign it to a variable.

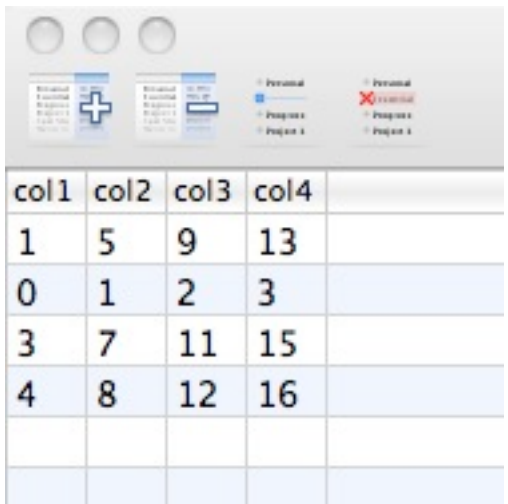


col1	col2	col3	col4
1	5	9	13
0	1	2	3
3	7	11	15
4	8	12	16

Matrices and vectors can be manipulated in a spreadsheet-like environment

```
> M6 <- edit(M1)
```

The command **`edit`** opens up the spreadsheet window, but in order for the changes you make in it to be saved you must assign it to a variable.



col1	col2	col3	col4
1	5	9	13
0	1	2	3
3	7	11	15
4	8	12	16

Now try Exercise 5

```
  > sum(MaqSqu[1,])
[1] 111
  > sum(MaqSqu[2,])
[1] 111
  > sum(MaqSqu[3,])
[1] 111
  > sum(MaqSqu[4,])
[1] 108
  > sum(MaqSqu[5,])
[1] 111
  > sum(MaqSqu[6,])
[1] 111
```

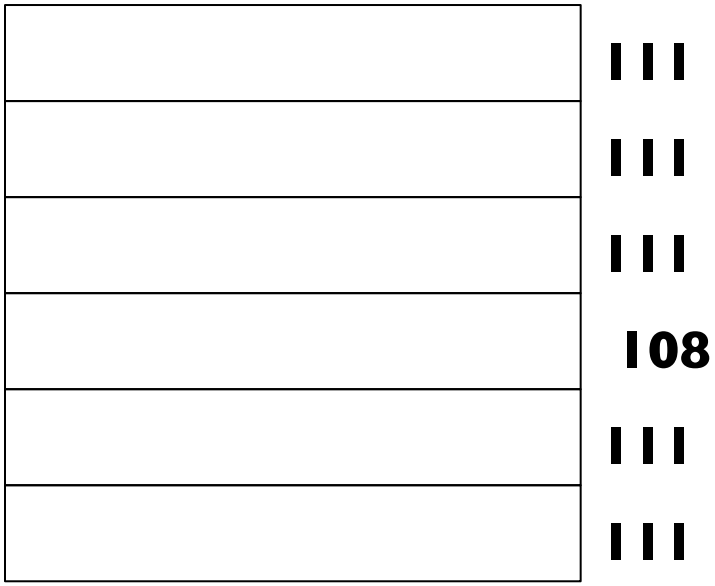
```
> sum(MaqSqu[1,])
[1] 111
> sum(MaqSqu[2,])
[1] 111
> sum(MaqSqu[3,])
[1] 111
> sum(MaqSqu[4,])
[1] 108
> sum(MaqSqu[5,])
[1] 111
> sum(MaqSqu[6,])
[1] 111
```

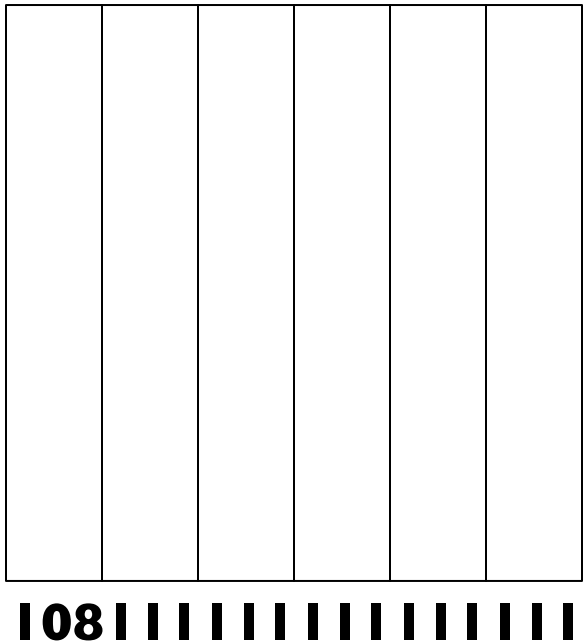
```
> sum(MaqSqu[,1])
[1] 108
> sum(MaqSqu[,2])
[1] 111
> sum(MaqSqu[,3])
[1] 111
> sum(MaqSqu[,4])
[1] 111
> sum(MaqSqu[,5])
[1] 111
> sum(MaqSqu[,6])
[1] 111
```

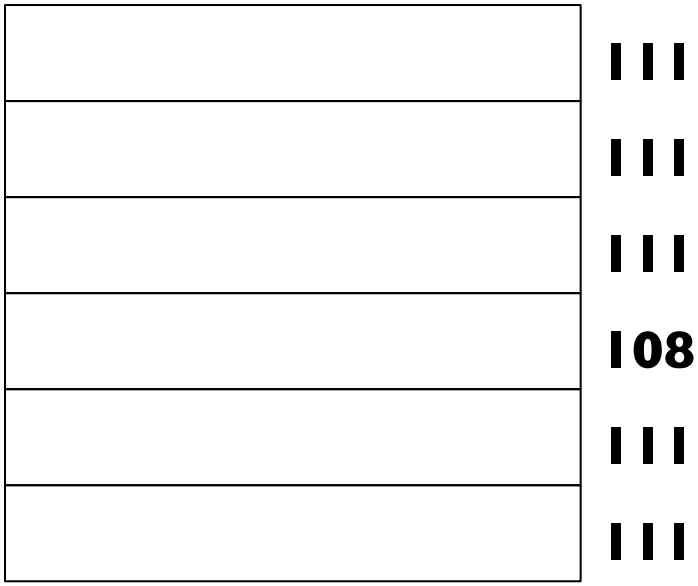
```
  > sum(MaqSqu[1,])
[1] 111
  > sum(MaqSqu[2,])
[1] 111
  > sum(MaqSqu[3,])
[1] 111
  > sum(MaqSqu[4,])
[1] 108
  > sum(MaqSqu[5,])
[1] 111
  > sum(MaqSqu[6,])
[1] 111
```

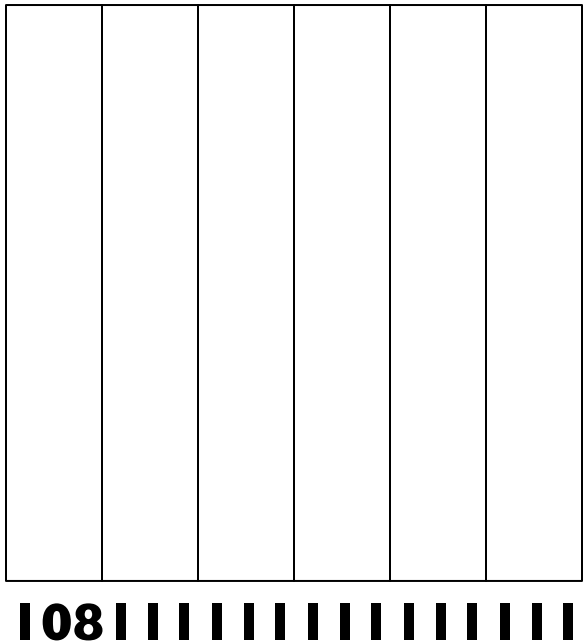
```
  > sum(MaqSqu[,1])
[1] 108
  > sum(MaqSqu[,2])
[1] 111
  > sum(MaqSqu[,3])
[1] 111
  > sum(MaqSqu[,4])
[1] 111
  > sum(MaqSqu[,5])
[1] 111
  > sum(MaqSqu[,6])
[1] 111
```

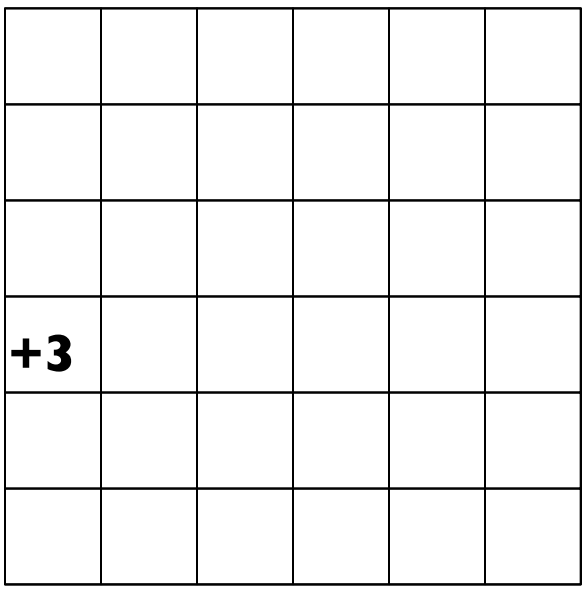
```
  > MaqSqu[1,1] + MaqSqu[2,2] + … + MaqSqu[6,6]
[1] 111
  > MaqSqu[1,6] + MaqSqu[2,5] + … + MaqSqu[1,6]
[1] 111
```












```
> MaqSqu[4,1]<-18
```

Matrix operations

```
> eigen(M1)
$values
[1]  3.1e+01 -2.1e+00  2.6e-15 -5.1e-16
$vectors
      [,1]  [,2]  [,3]  [,4]
[1,] -0.47  0.87 -0.54  0.23
[2,] -0.10  0.32  0.69  0.04
[3,] -0.58  0.04  0.25 -0.81
[4,] -0.64 -0.36 -0.39  0.52
```

R can calculate the eigenvalues and eigenvectors of a matrix.

Matrix operations

Matrix multiplication

```
> M1 * M2
     [,1] [,2] [,3] [,4]
[1,]    1   10   27   52
[2,]    0    6   14   24
[3,]   27   70  121  180
[4,]   52  112  180  256
```

Matrix-vector multiplication

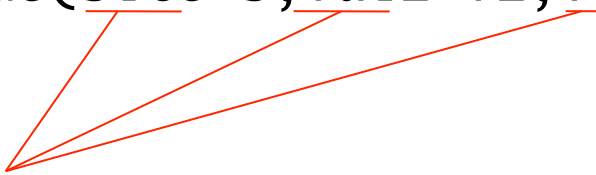
```
> M3 %*% v
   [,1]
w1  125
w5   60
v    55
```

Transpose

```
> M5 <- t(M1)
```

Data frames are useful data structure, they function like a matrix with labeled columns. But unlike a matrix they can contain a mixture of columns of numbers and columns of characters.

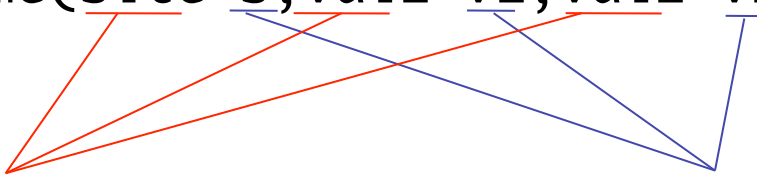
```
> mydata <- data.frame(site=s,val1=v1,val2=v2)
```



For `data.frame` the arguments are not set, like with other functions we have seen, but are chosen by you as the names of your columns.

Data frames are useful data structure, they function like a matrix with labeled columns. But unlike a matrix they can contain a mixture of columns of numbers and columns of characters.

```
> mydata <- data.frame(site=s,val1=v1,val2=v2)
```



For `data.frame` the arguments are not set, like with other functions we have seen, but are chosen by you as the names of your columns.

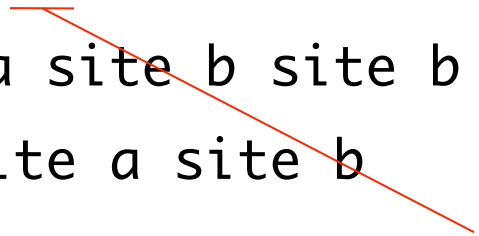
The values you set for these arguments are your vectors of data.

Data frames are useful data structure, they function like a matrix with labeled columns. But unlike a matrix they can contain a mixture of columns of numbers and columns of characters.

```
> mydata <- data.frame(site=s,val1=v1,val2=v2)
> mydata
  site val1 val2
1 site a    3    4
2 site b    6    3
3 site b    4    7
4 site a   27  -14
5 site a   -2   17
```

Indexing data frames

```
> mydata$site
[1] site a site b site b site a site a
Levels: site a site b
```



The $ is used to refer to a given column of a data frame.

Indexing data frames

```
  > mydata$site
[1] site a site b site b site a site a
Levels: site a site b
  > mydata$val1[1:2]
[1] 4 27
  > mydata$val2[mydata$site == “site b”]
[1] 3 7
```

From there we can index the columns of the data frame just like we did vectors.

Getting your data in from excel.

Need to save excel spreadsheet as a .csv. Go to File, Save As, and then change Format to csv.

- will save the values, not equations

- will save just one sheet

- any non-numeric turns a whole column into character mode, even if the rest of the entries are numbers. Blank values are accepted.

- each column must be named

- R will turn certain characters (*, /, spaces, parentheses) in column names into periods (.)

Now we are ready to get it into R

```
> bwdata <- read.csv(file='big_woods.csv')
```

Now we are ready to get it into R

```
> bwdata <- read.csv(file='big_woods.csv')
```

By default R loads the data from the .csv in as a data frame, so `bwdata` is a data frame. To see what the column names are use the command `names`

```
> names(bwdata)
[1] "Species" "x" "y" "gbh03" "gbh08" "notes"
```

Now we are ready to get it into R

```
> bwdata <- read.csv(file='big_woods.csv')
```

By default R loads the data from the .csv in as a data frame, so `bwdata` is a data frame. To see what the column names are use the command `names`

```
> names(bwdata)
[1] "Species" "x" "y" "gbh03" "gbh08" "notes"
```

We can call these values just like we saw with other data frames.

```
> bwdata$x[1:5]
[1] 153.7 146.4 138.0 150.0 144.4
> bwdata$species[1:5]
NULL
```

You have to get the names *exactly* right. R is case sensitive.

Typing out the `bwdata$` every time gets annoying. Use the `attach` command to make R assume you typed `bwdata$`.

```
> attach(bwdata)
> x[1:5]
[1] 153.7 146.4 138.0 150.0 144.4
```

Typing out the **bwdata$** every time gets annoying. Use the **attach** command to make R assume you typed **bwdata$**.

```
> attach(bwdata)
> x[1:5]
[1] 153.7 146.4 138.0 150.0 144.4
```

You can attach more than one data frame, but if they have columns with the same name you can get in trouble.

```
> swdata <- read.csv(file='small_woods.csv')
> attach(swdata)
The following object(s) are masked from 'bwdata':
    Species x y gbh03 gbh08 notes
```

This means that the **x** from **swdata** 'masks' the **x** of **bwdata**. So when you type just **x** you get the **x**es from **swdata** not **bwdata**. Similarly for the other variables.

Use the `search` command to see what you have attached and the order that R will look at them to decide what value to give you if there is a conflict.

```
> search()
[1] ".GlobalEnv"  "swdata"   "bwdata" ...
```

There will be other `tool`s and the `package`s after but we can ignore them for now. The important point here is that R is looking at `swdata` first and then `bwdata`. R 'looks' at more recently `attach`ed data frames first.

Use the `search` command to see what you have attached and the order that R will look at them to decide what value to give you if there is a conflict.

```
> search()
[1] ".GlobalEnv"  "swdata"    "bwdata" ...
```

Use the `detach` command to remove a data frame from R's search path.

```
> detach(swdata)
> search()
[1] ".GlobalEnv"  "bwdata" ...
> x[1:5]
[1] 153.7 146.4 138.0 150.0 144.4
```

Since we have detached `swdata` now when we just type `x` we get `bwdata`'s `x`es. `swdata`'s `x`es are still 'there' and you can access them using `swdata$x`, they have just been removed from R's search path.

We assume that most of you will want to important your data into R from Excel files. There can be some stumbling blocks when doing this. These are exposed on the Exercise 6.

Change 'missing data' to blank, so that R will read that column as numeric.

Add a column header to column E on the 'green frogs' sheet.

Change headers so that R gives you reasonable names.

```
> green_frog <- read.csv('greenfrog.csv')
> wood_frog <- read.csv('woodfrog.csv')
> names(green_frog)
[1] "density"  "predator"   "tadpole.size"
[4] "number_surviving" "notes"
> attach(green_frog)
> mean(number_surviving[predator=='pred']/density
 [predator=='pred'])
[1] 0.5304762
> detach(green_frog)
> attach(wood_frog)
```

Now that you have matrices, vectors, and data frames you might want to save them for later

```
> save(file='R_session_Aug31.Rdata',bwdata,M1,v)
```

And later to load it back

```
> load(file='R_session_Aug31.Rdata')
```

Data analysis and graphing

basic descriptive stats

simple tests

plots

histograms

Descriptive statistics

```
> mean(gbh03[Species == 'Black Cherry'])
```

`sd, quantile, var, …`

These commands work like `length`, `sum`, … on vectors.

```
> cov(v1,v2)
```

Find the covariance between two vectors

Linear regression, say you have two vectors `height` and `seed_mass`*

```
> mymodel <- lm(formula = seed_mass ~ height)
```

* These data are completely made up.

Linear regression, say you have two vectors `height` and `seed_mass`*

```
> mymodel <- lm(formula = seed_mass ~ height)
> summary(mymodel)
```

```
Call:
lm(formula = seed_mass ~ height)

Residuals:
    Min      1Q  Median      3Q     Max
-6.2507 -2.0307 -0.4568  1.5298 10.5006

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   6.3466     0.6185  10.261  < 2e-16 ***
height        2.9163     0.9497   3.071  0.00276 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.249 on 98 degrees of freedom
Multiple R-squared: 0.08777,    Adjusted R-squared: 0.07846
F-statistic: 9.429 on 1 and 98 DF,  p-value: 0.002765
```

* These data are completely made up.

Linear regression, say you have two vectors `height` and `seed_mass`*

```
> mymodel <- lm(formula = seed_mass ~ height)
> summary(mymodel)
```

```
Call:
lm(formula = seed_mass ~ height)

Residuals:
    Min      1Q  Median      3Q     Max
-6.2507 -2.0307 -0.4568  1.5298 10.5006

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   6.3466     0.6185  10.261  < 2e-16 ***
height        2.9163     0.9497   3.071  0.00276 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.249 on 98 degrees of freedom
Multiple R-squared: 0.08777,    Adjusted R-squared: 0.07846
F-statistic: 9.429 on 1 and 98 DF,  p-value: 0.002765
```

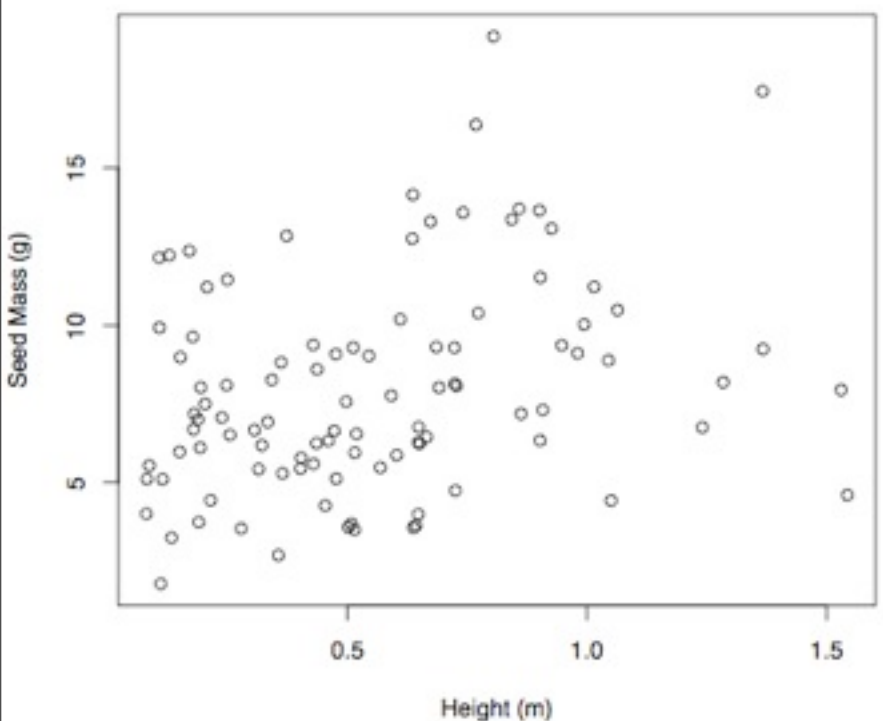
For general linear models use `glm.`

Many statistical tests `t.test`, `anova`, `wilcox.test`, ...

* These data are completely made up.

Simple plots

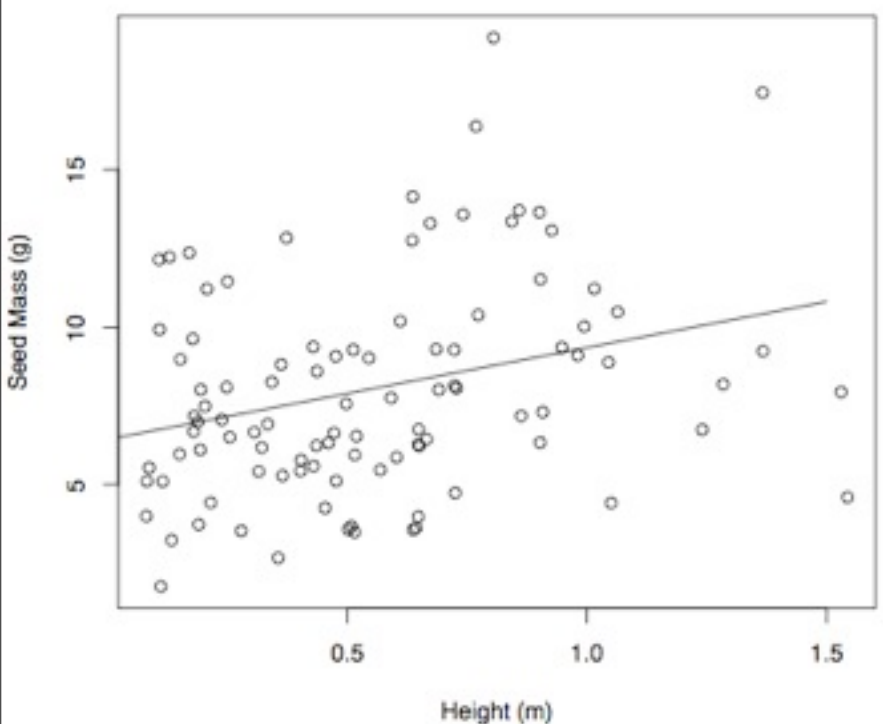
```
> plot(x=height,
       y=seed_mass,
       xlab='Plant Height (m)',
       ylab='Seed Mass (g)' )
```



The `plot` creates a new plot, covering over any existing one you have, and plots the values you have in x against those in y.

Adding a line to our plot. Lines are defined by as few as two points. In this case using the coefficients of our linear model.

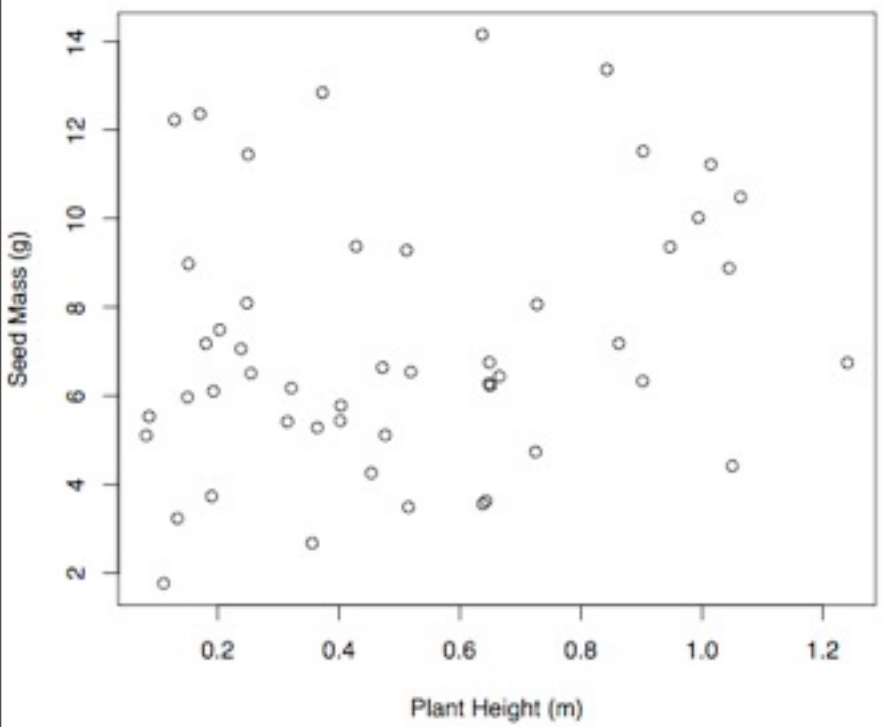
```
> lines(x=c(0,1.5),
        y=c(6.437,6.437+2.916*1.5))
```



The command line does not call a new plot and erase the current one, but plots onto the current one.

Now let's say the plants came from two different sites and you want to indicate those two different sites, and that information is stored in `site`.

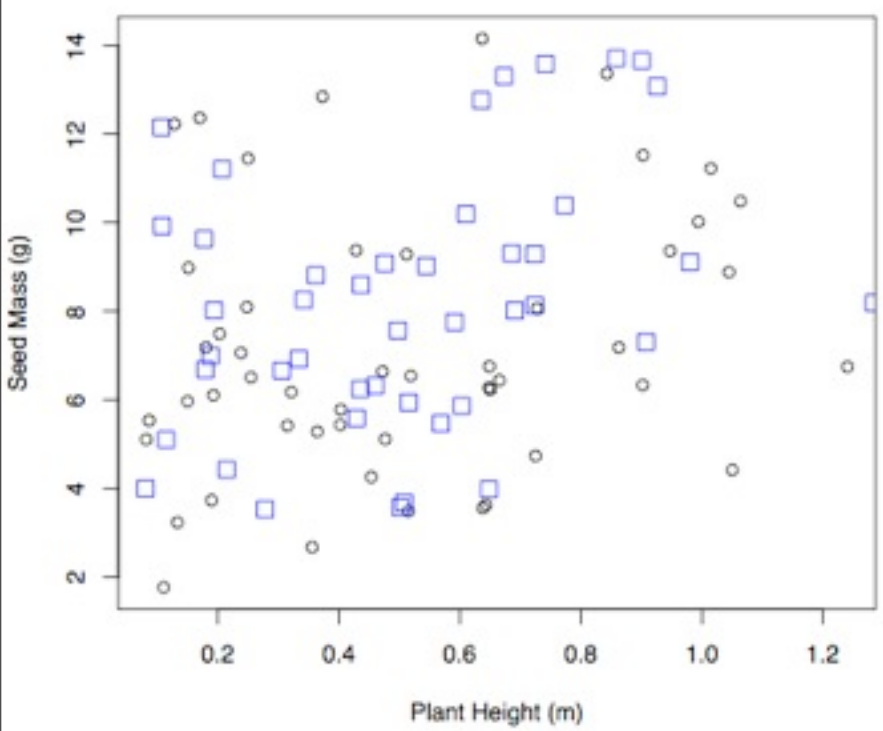
```
> plot(x=height[site== 'site 1'],
       y=seed_mass[site== 'site 1'],
       xlab='Plant Height (m)',
       ylab='Seed Mass (g)' )
```



Now let's say the plants came from two different sites and you want to indicate those two different sites, and that information is stored in `site`.

```
> points(x=height[site== 'site 2'],
         y=seed_mass[site== 'site 2],
          pch=0,
         col='blue', cex=1.5)
```

`points`, like `lines`, adds onto an existing plot



`pch` determines the shape plotted. The numbers run from 0 to 19, the default is 1 (open circles).

`col` the color. Here you can name most of the standard colors (remember to use quotation marks), or define any color using the command `rgb.`

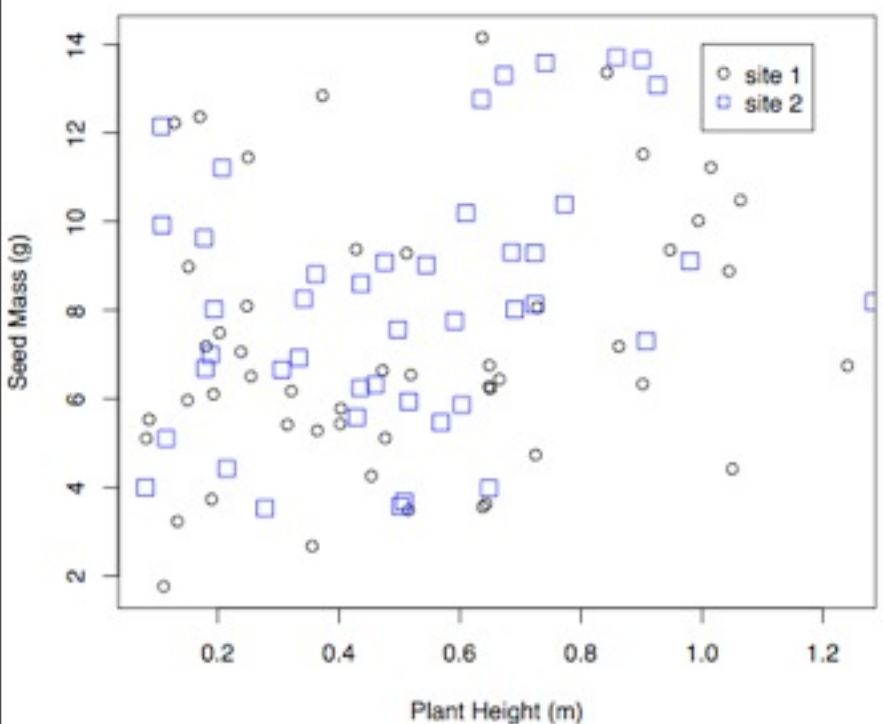
`cex` controls the size of the points, with default at 1.

Now let's say the plants came from two different sites and you want to indicate those two different sites, and that information is stored in `site`.

```
> legend(x=1,y=14,
        legend=c('site 1','site 2'),
       pch=c(1,4),
       col=c('black','blue') )
```

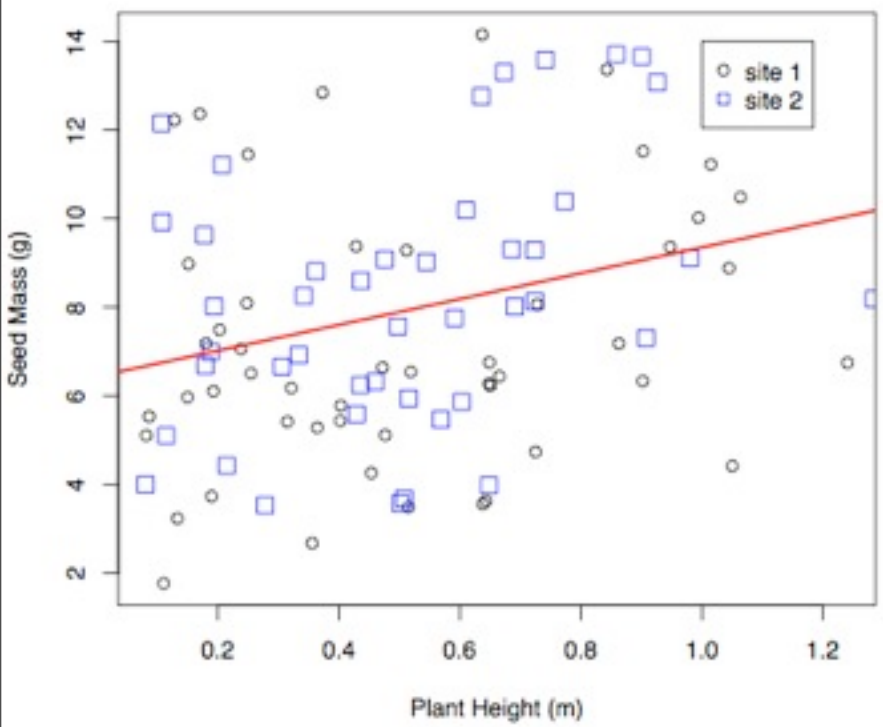
These set the upper-left point of the legend

Since you are giving R more than one value for these arguments remember to use the `c` command.



Now let's say the plants came from two different sites and you want to indicate those two different sites, and that information is stored in `site`.

```
> lines(x=c(0,1.5),
        y=c(5.035,5.035+1.979*1.5),
        col='red',
        lwd=2)
```



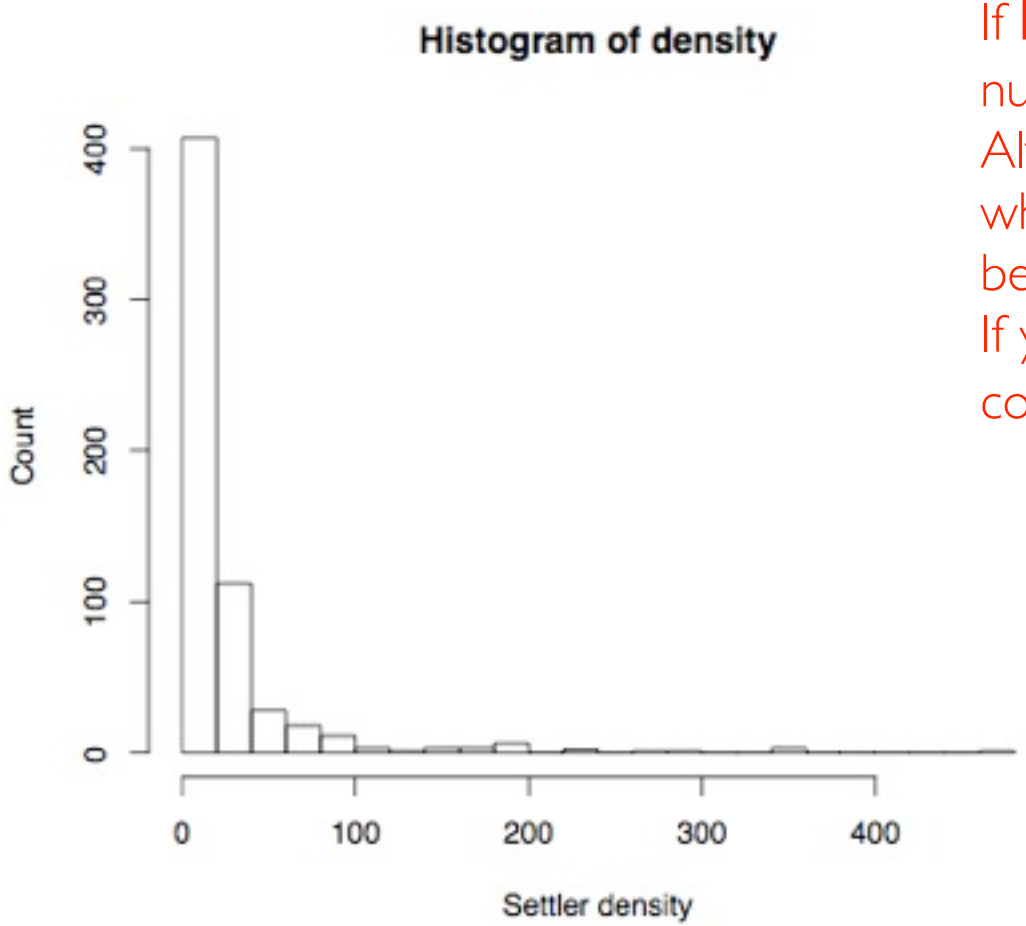
Histograms, here we have a vector of densities of damsel fish that settle in an area called `density`*

```
> denhist <- hist(x=density,breaks=25,
                  xlab="Settler Density")
```

* Data for this example from Schmitt et al. (1999), as made available through Bolker (2008) in the data set `DamselSettlement`

Histograms, here we have a vector of densities of damsel fish that settle in an area called `density`*

```
> denhist <- hist(x=density,breaks=25,
                  xlab="Settler Density")
```



If `breaks` is a number it sets the number of cells for the histogram. Alternatively you can give a vector, which will be the breakpoints between the histogram cells.
If you want frequencies rather than counts add `freq=TRUE`.

* Data for this example from Schmitt et al. (1999), as made available through Bolker (2008) in the data set `DamselSettlement`

Histograms, here we have a vector of densities of damsel fish that settle in an area called `density`*

```
> denhist <- hist(x=density,breaks=25,
                  xlab="Settler Density")
```

In addition to the plot we also have the object `denhist`, which has all of the information from the histogram.

```
> denhist$mids
[1] 10 30 50 70 90 …
> denhist$counts
[1] 407 112 28 18 11 …
```

* Data for this example from Schmitt et al. (1999), as made available through Bolker (2008) in the data set `DamselSettlement`

Graphs will appear in a window, but can also be saved number of formats

```
> jpeg(file='SpNum_Den.jpeg',width=500,height=500)
> plot(x=density,y=sp_num,…)
> lines( …)
> points(…)
> legend(…)
> dev.off()
```

pdf, tiff, png, …

`jpeg` (or `pdf`, `png`, …) tells R to start creating a new image and then once R sees `dev.off()` the image is locked in.

Additional graphical control using the **par** command.

```
> par(cex.lab=1.5,bty='n',mfrow=c(1,2))
```

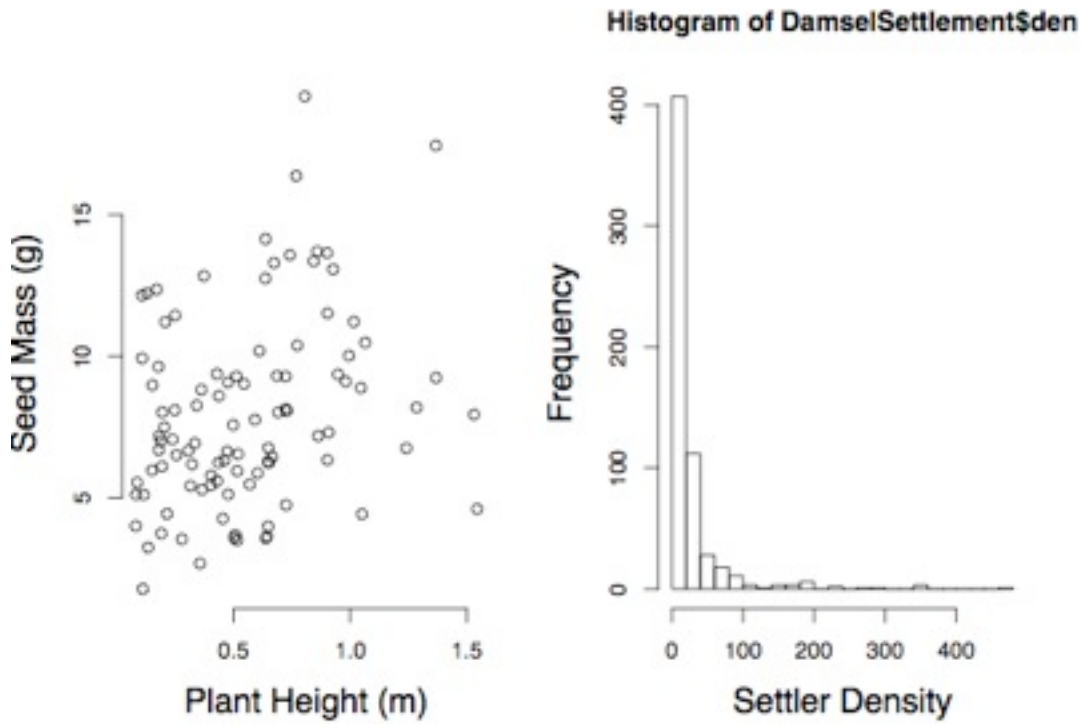
Sets the font size for the axis labels. Similarly **cex.axis** and **cex.main**, which set the font size of axis numbers and the main title.

Removes the box around the plot.

Tells R you want a graph with multiple plots. In this case one row and two columns of plots, so two plots.

Additional graphical control using the `par` command.

```
> par(cex.lab=1.5,bty='n',mfrow=c(1,2))
> plot(height,seed_mass,xlab="Plant Height (m)",
      ylab="Seed Mass (g)" )
> hist(x=density,breaks=25,xlab="Settler Density")
```

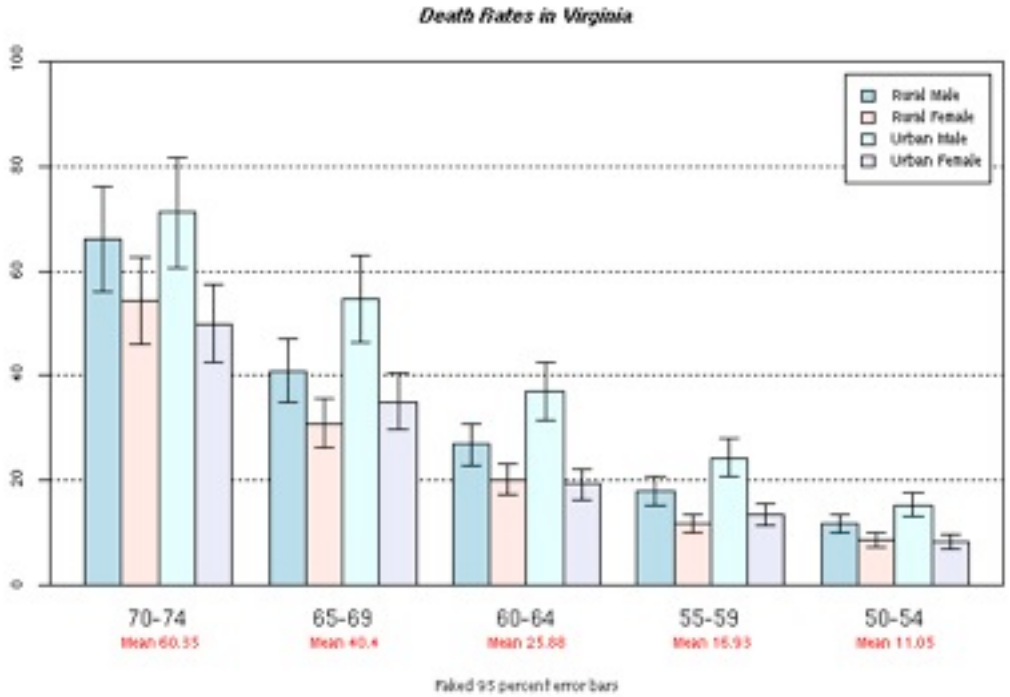


Tons and tons of other graphical parameters controlled by `par`, to see the list type `?par`

This is just scratching the surface, you can make very sophisticated graphs in R.

You can see a broad sampling of such graphs at: `http://addictedtor.free.fr/graphiques`

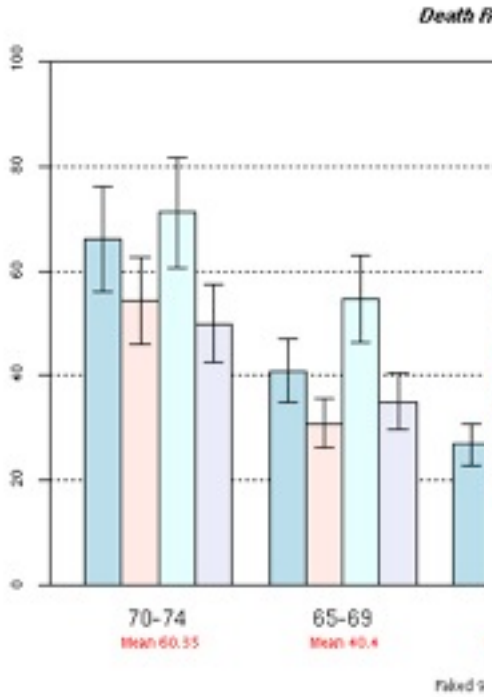
The site has a gallery of graphs and includes source code for each.



This is just scratching the surface, you can make very sophisticated graphs in R.

You can see a broad sampling of such graphs at: `http://addictedtor.free.fr/graphiques`

The site has a gallery of graphs and includes source code for each.

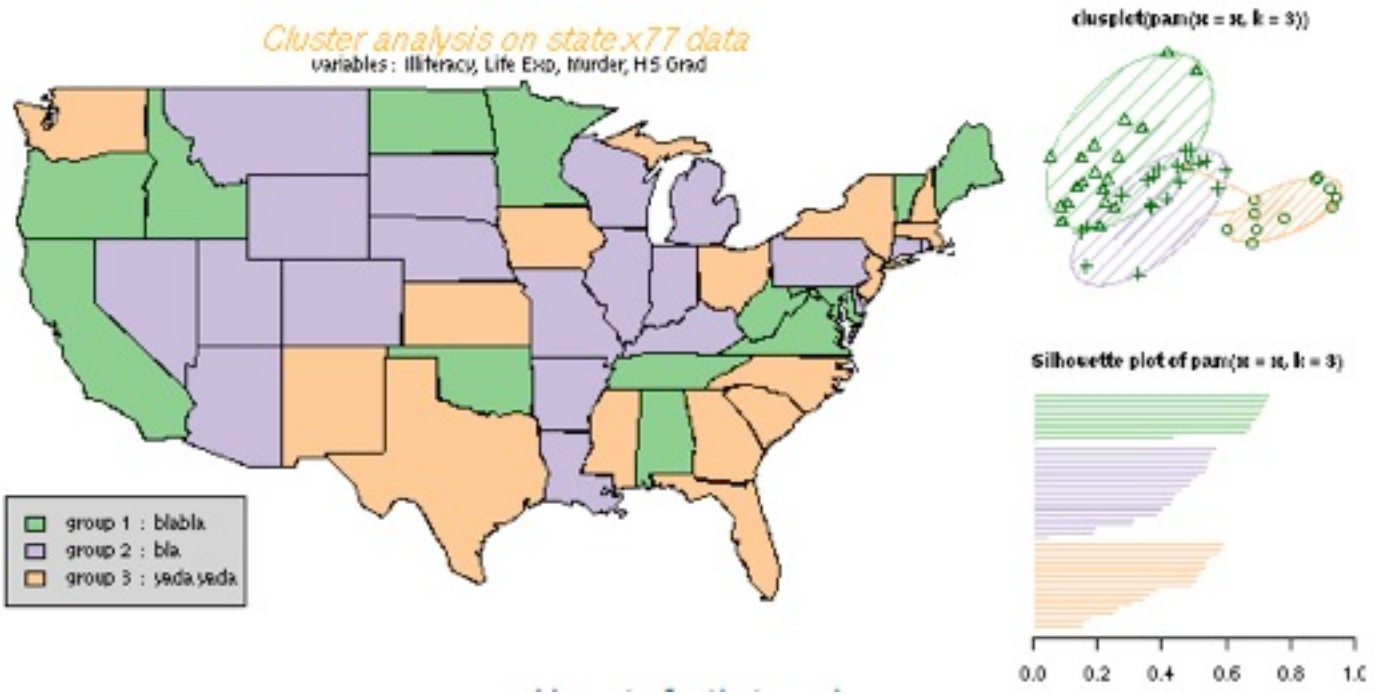



```
library(gplots)
# Example with confidence intervals and grid
hh <- t(VADeaths)[, 5:1]
mybarcol <- "gray20"
ci.l <- hh * 0.85
ci.u <- hh * 1.15
mp <- barplot2(hh, beside = TRUE,
        col = c("lightblue", "mistyrose",
                "lightcyan", "lavender"),
        legend = colnames(VADeaths), ylim = c(0, 100),
        main = "Death Rates in Virginia", font.main = 4,
        sub = "Faked 95 percent error bars", col.sub = mybarcol,
        cex.names = 1.5, plot.ci = TRUE, ci.l = ci.l, ci.u = ci.u,
        plot.grid = TRUE)
mtext(side = 1, at = colMeans(mp), line = 2,
      text = paste("Mean", formatC(colMeans(hh))), col = "red")
box()
```

This is just scratching the surface, you can make very sophisticated graphs in R.

You can see a broad sampling of such graphs at: `http://addictedtor.free.fr/graphiques`

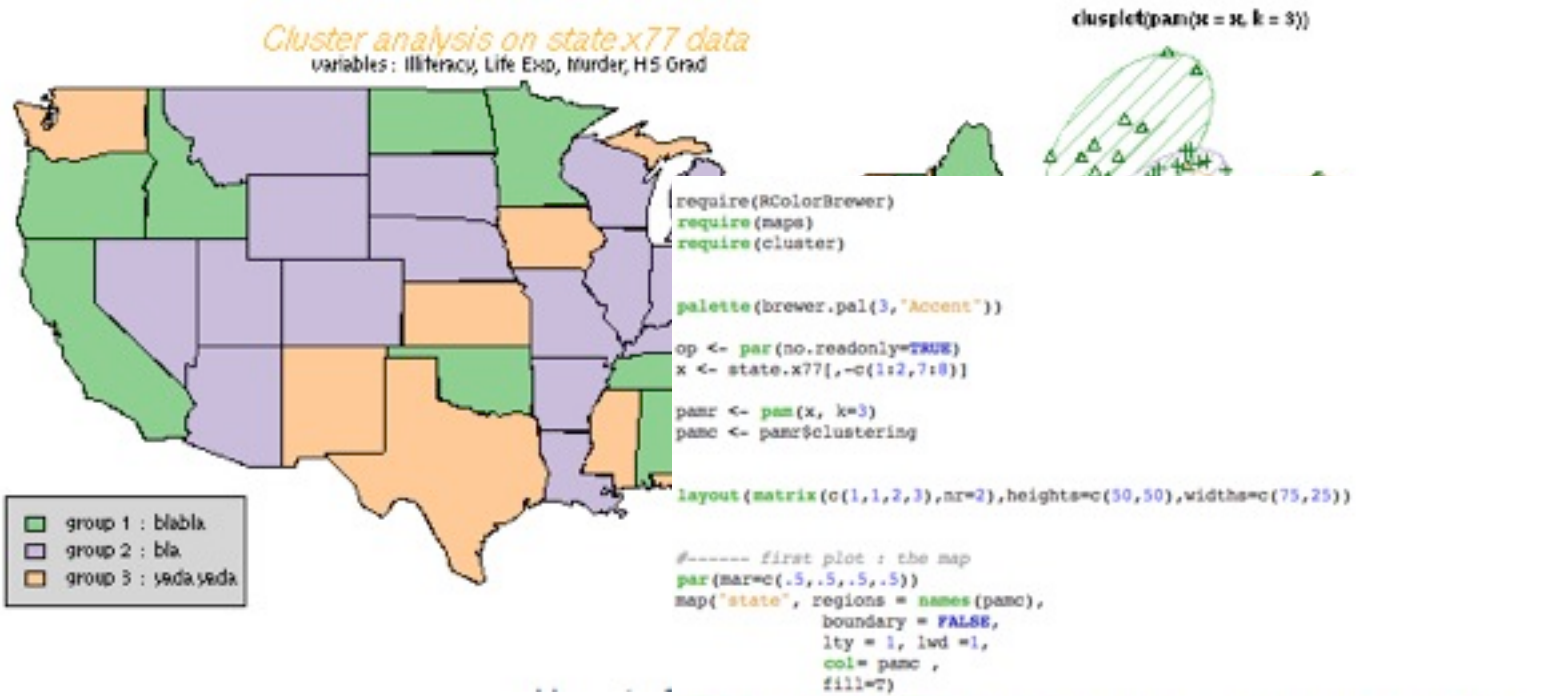
The site has a gallery of graphs and includes source code for each.



This is just scratching the surface, you can make very sophisticated graphs in R.

You can see a broad sampling of such graphs at: `http://addictedtor.free.fr/graphiques`

The site has a gallery of graphs and includes source code for each.



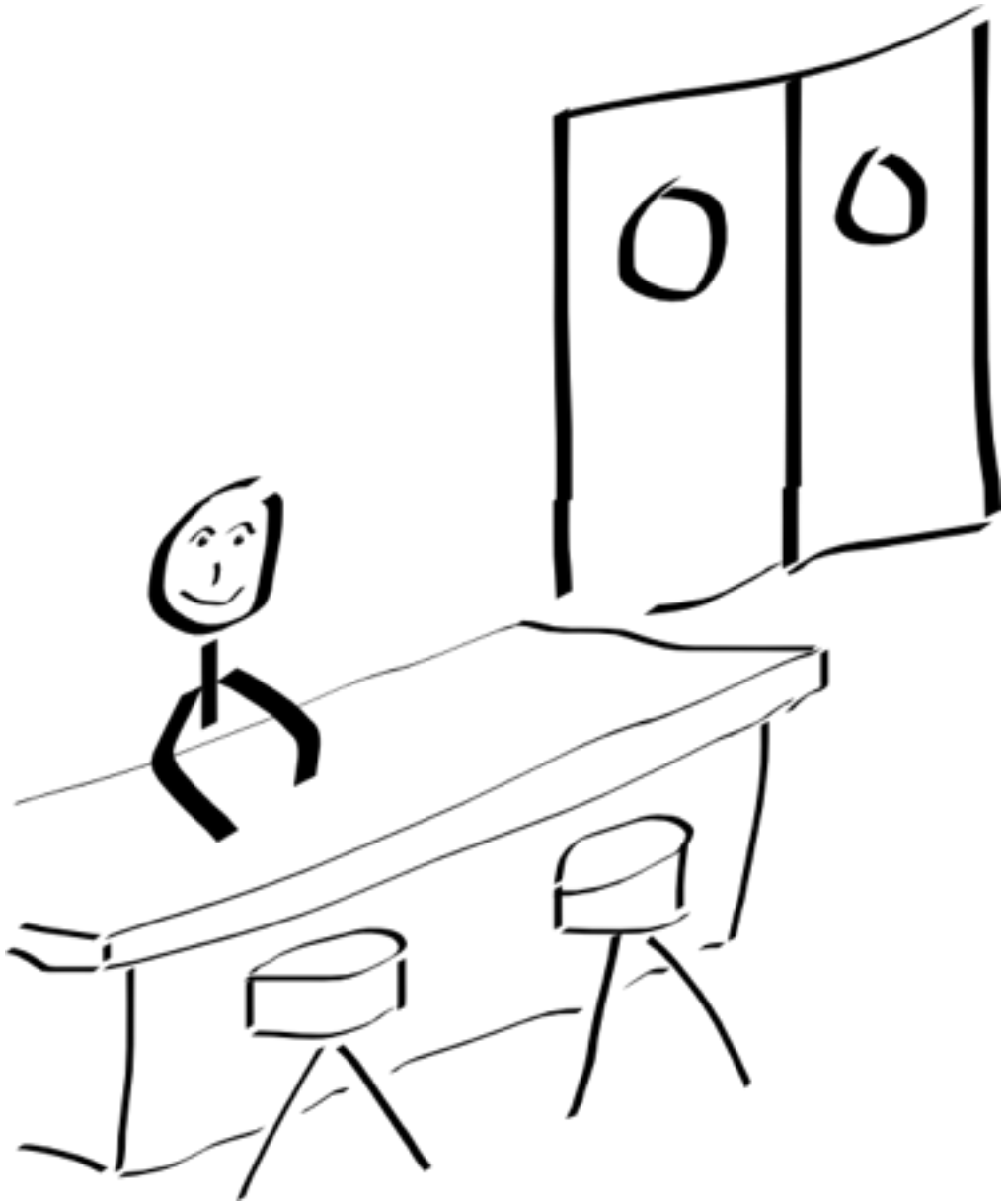
Simple data analysis and graphing are explored in exercise 7.

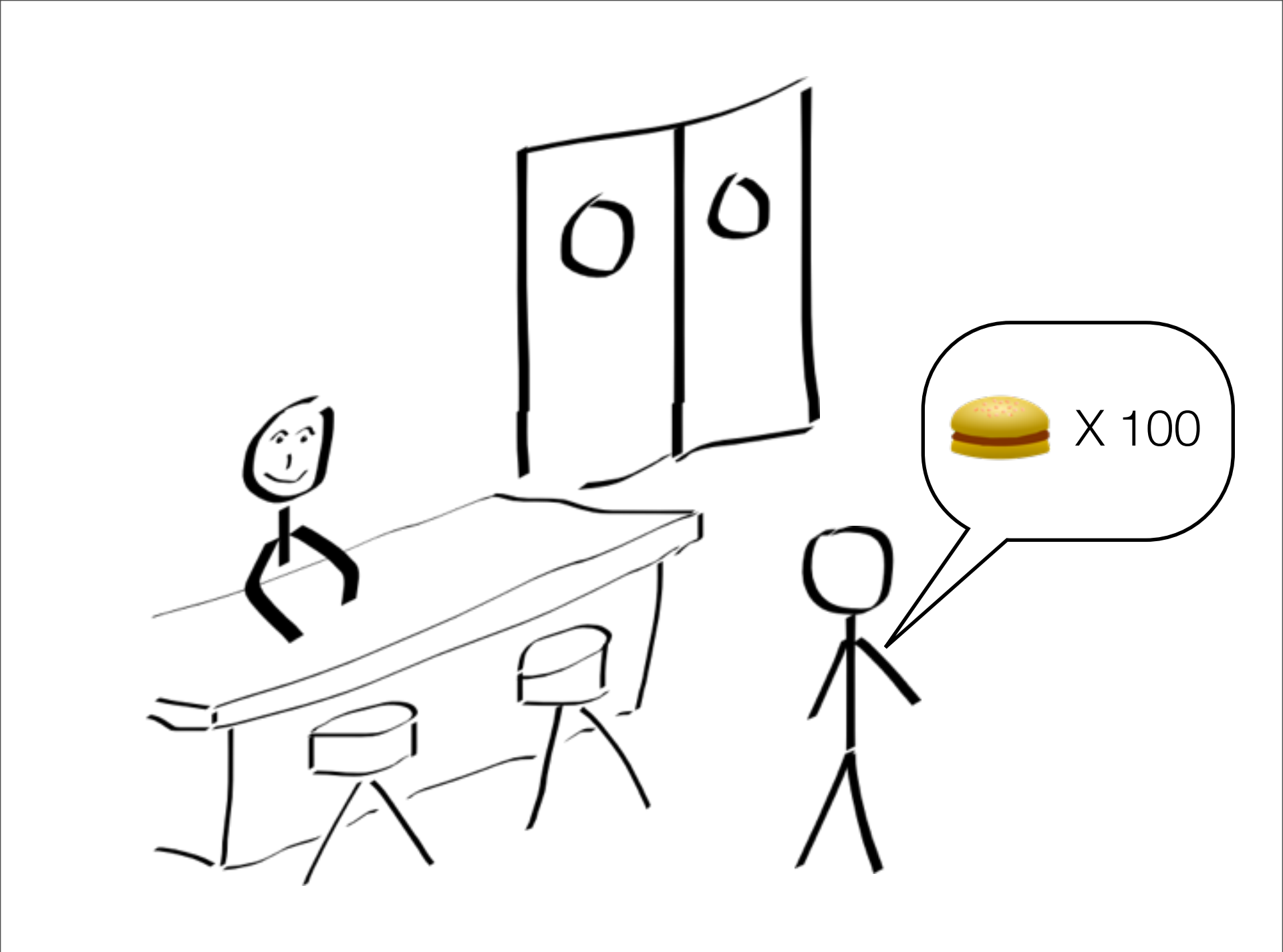
Programming in R

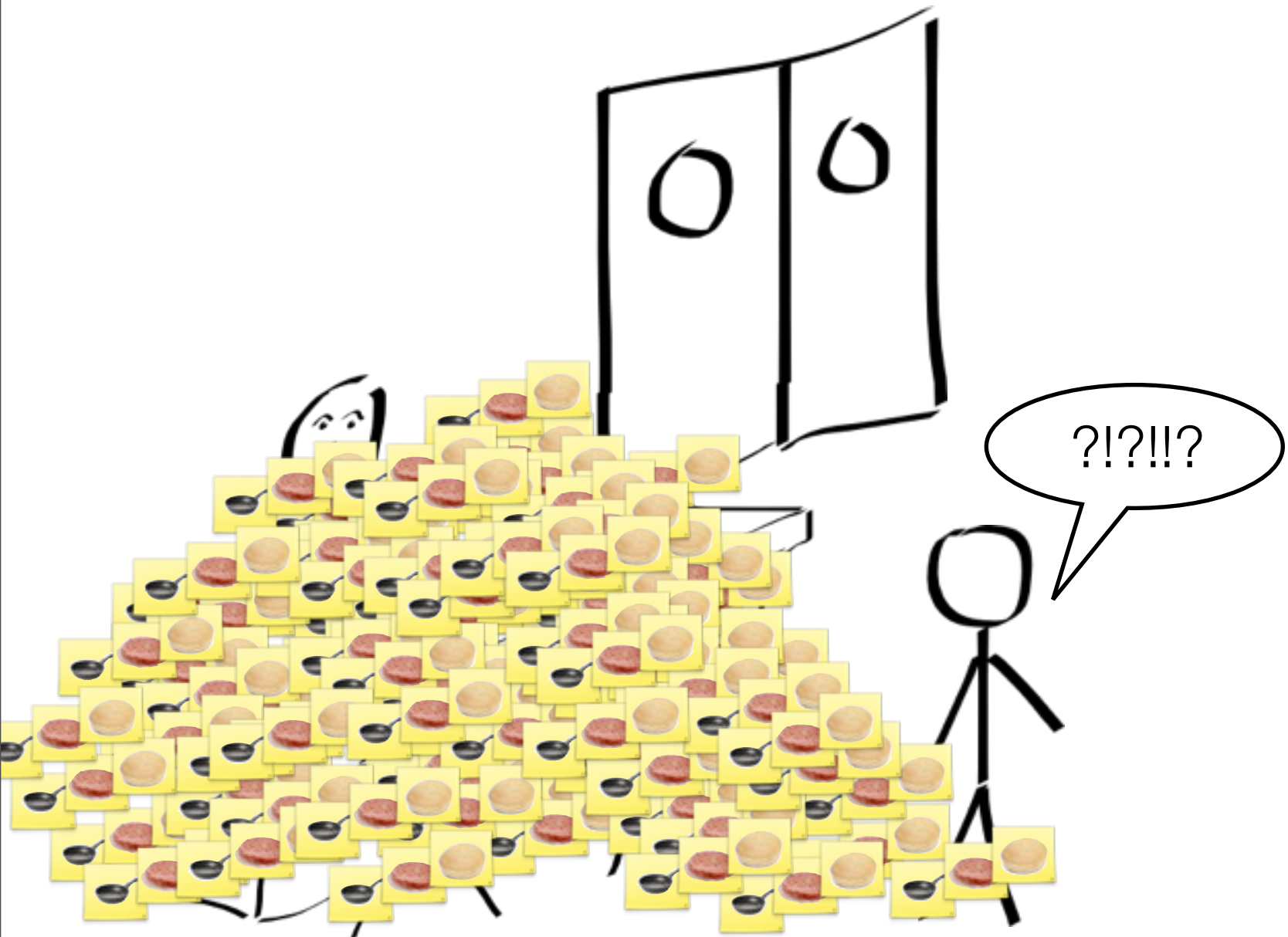
Doug Jackson and Dave Allen
31-Aug-2011

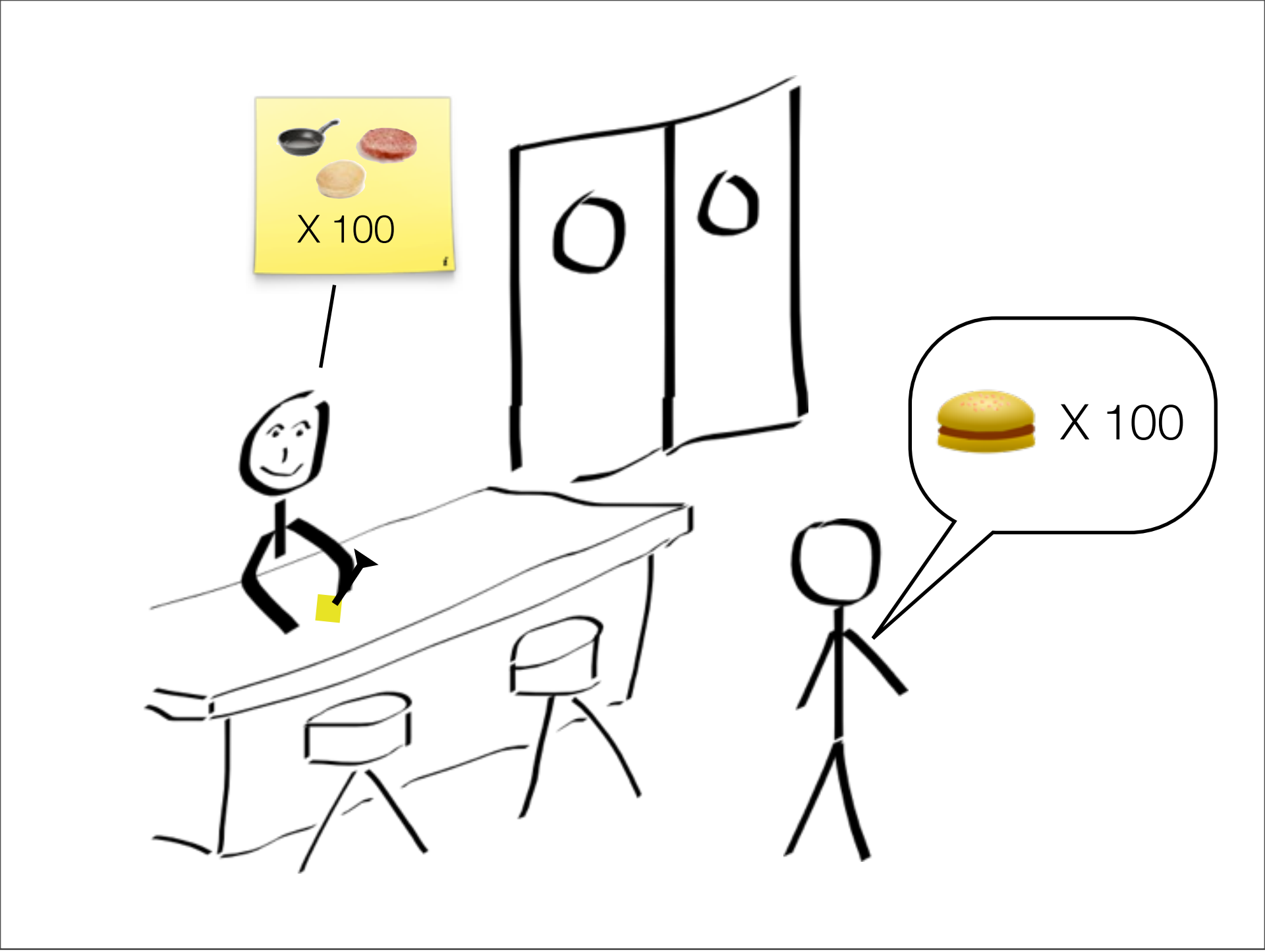
“Programming”


FEEL GOOD
DINER
Coca-Cola
Coca-Cola




X 100



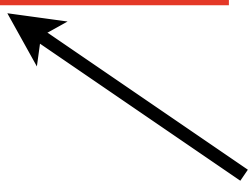

X 100
X 100

Control structures: `for` loops

```
# always comment your code!

# I guarantee you'll thank yourself later

for(i in 1:100)

{

        do stuff

}
```

Control structures: `for` loops

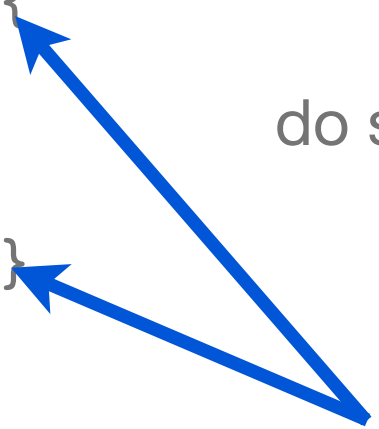
```
# do stuff 100 times

for(i in 1:100)

{

        do stuff

}
```



i will be incremented
each time through the loop,
starting at 1 and ending at 100

Control structures: `for` loops

```
# do stuff 100 times

for(i in 1:100)

{

}
```

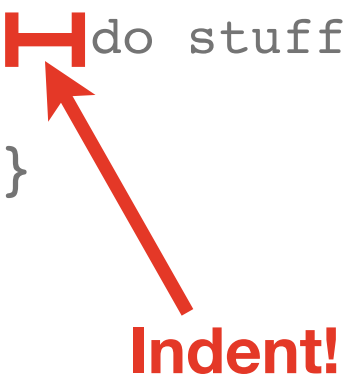


do stuff

Important!
Braces mark beginning and end of loop

Control structures: `for` loops

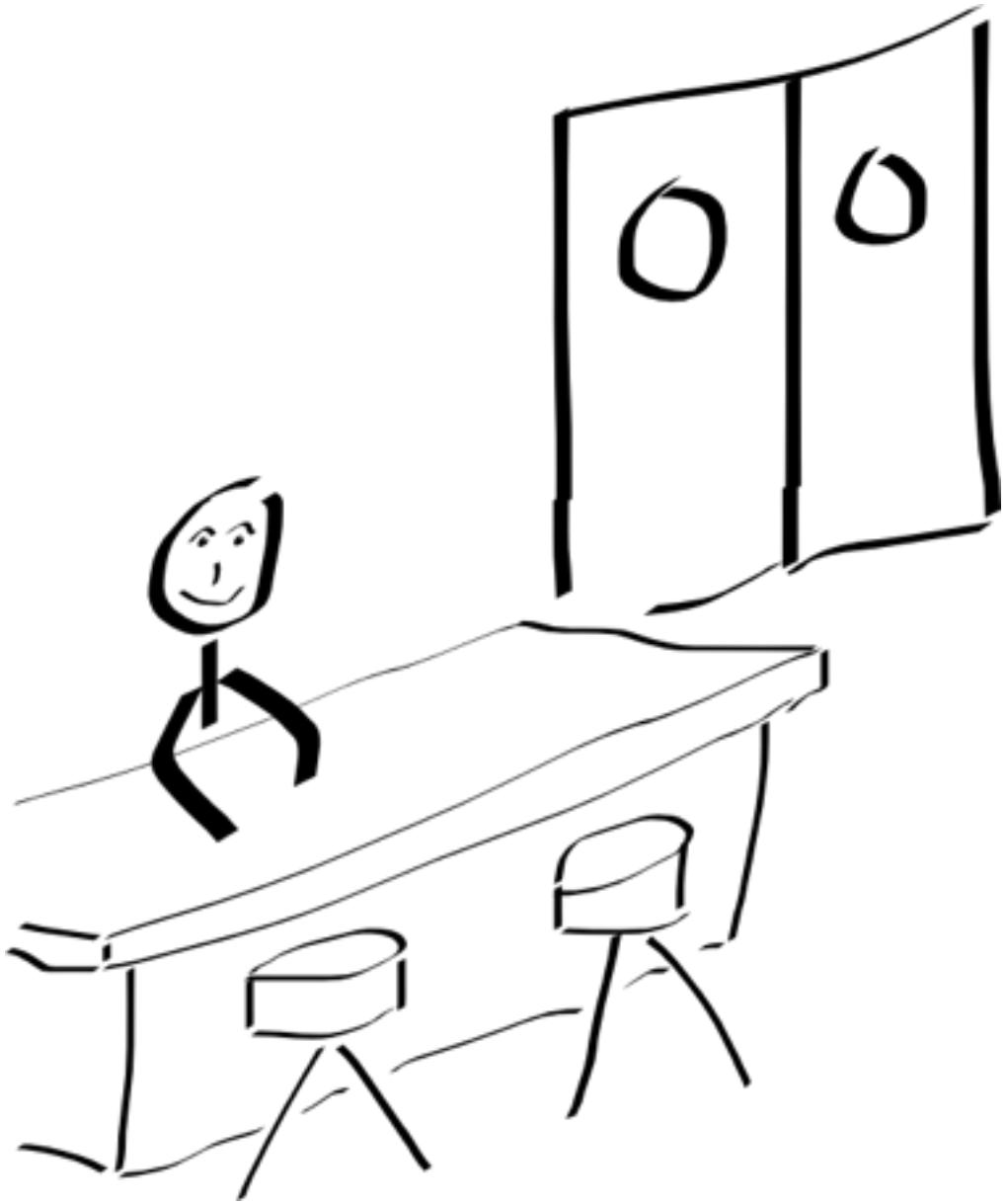
```
# do stuff 100 times

for(i in 1:100)

{

    do stuff

}
```

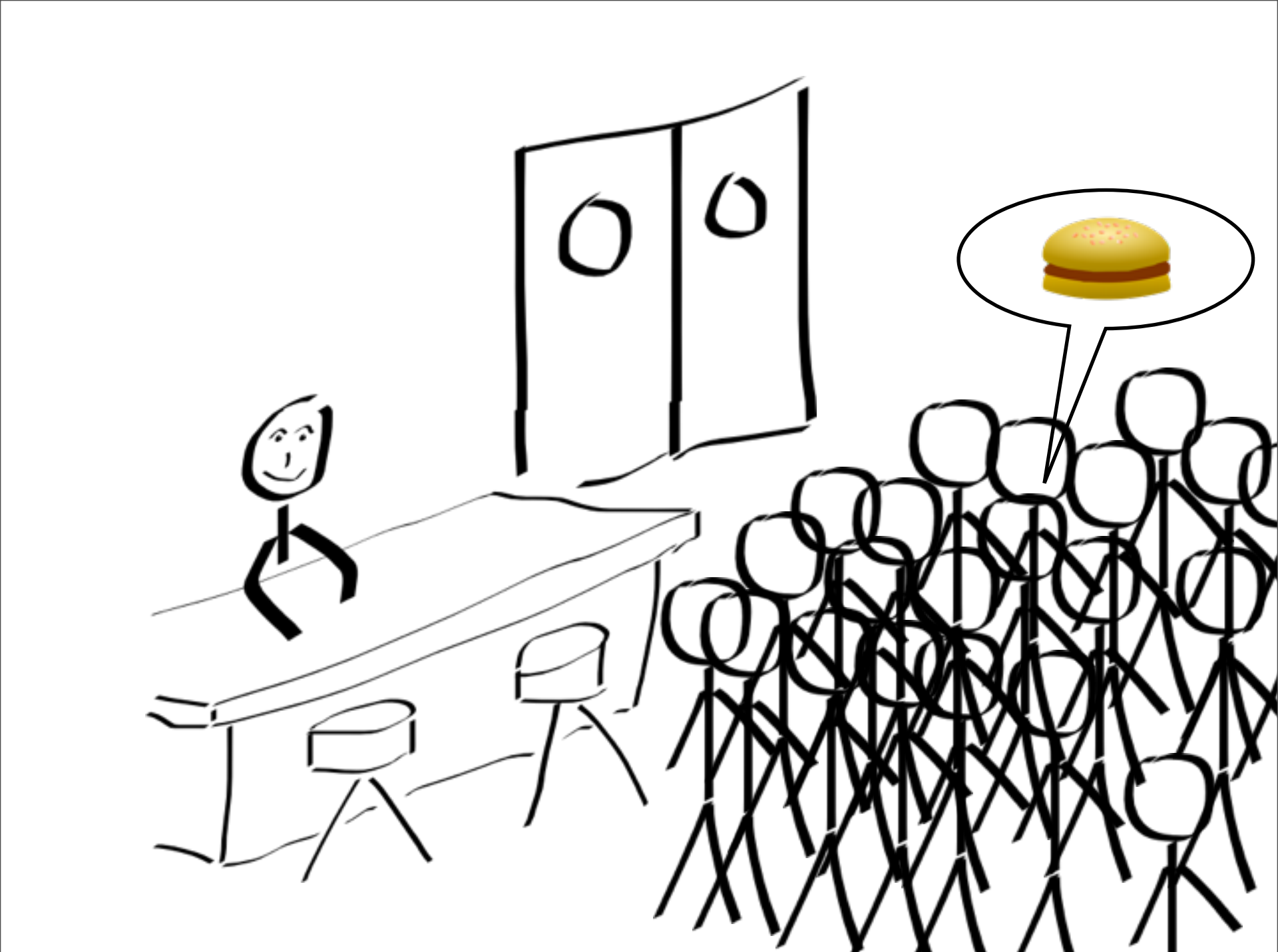


Control structures: `for` loops example

```
# sum all numbers from 1 to 100

cumulative_sum <- 0

for(i in 1:100)

{

   cumulative_sum <- cumulative_sum + i

}

print(cumulative_sum)
```

Exercise 8





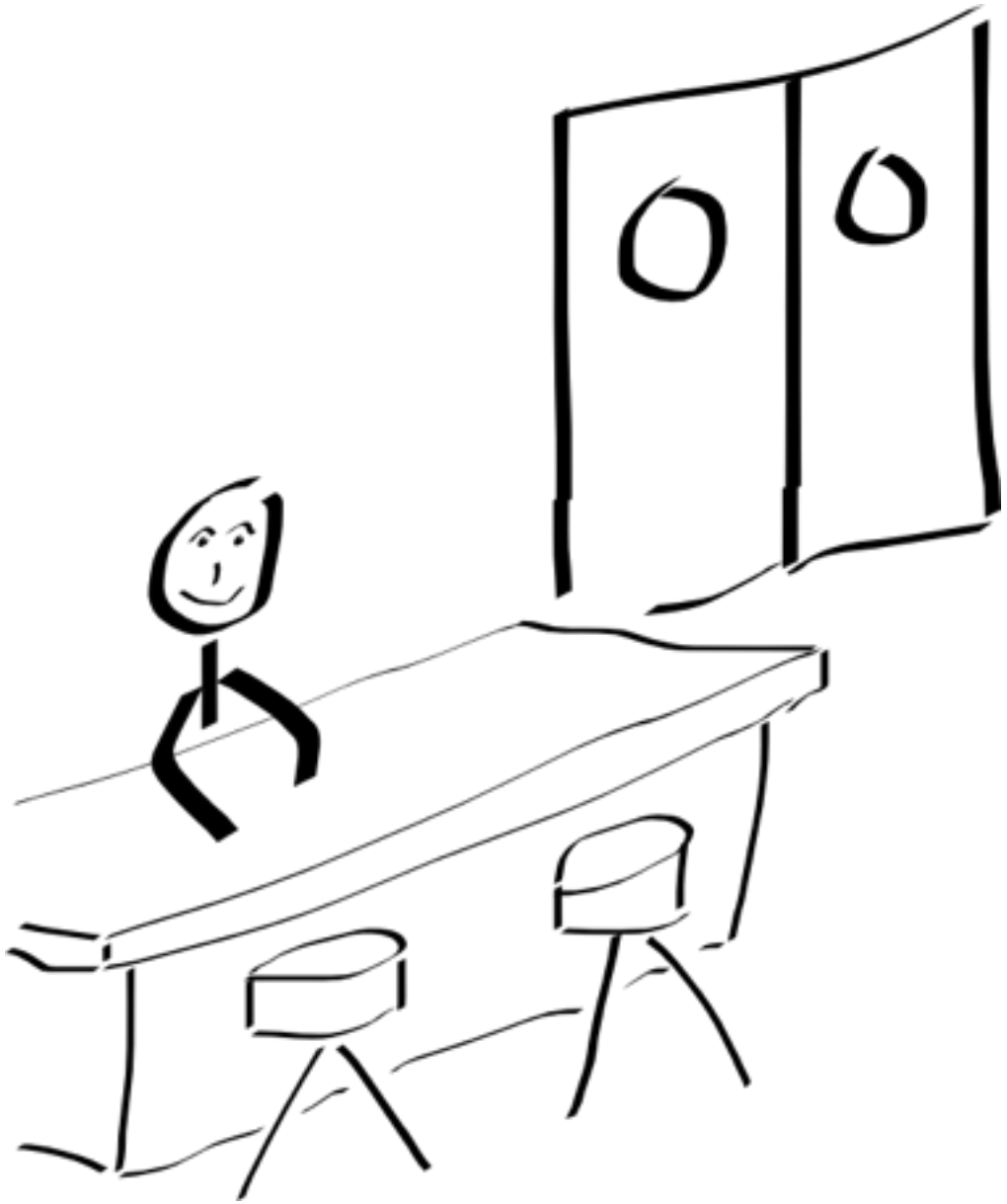
Control structures: `while` loops

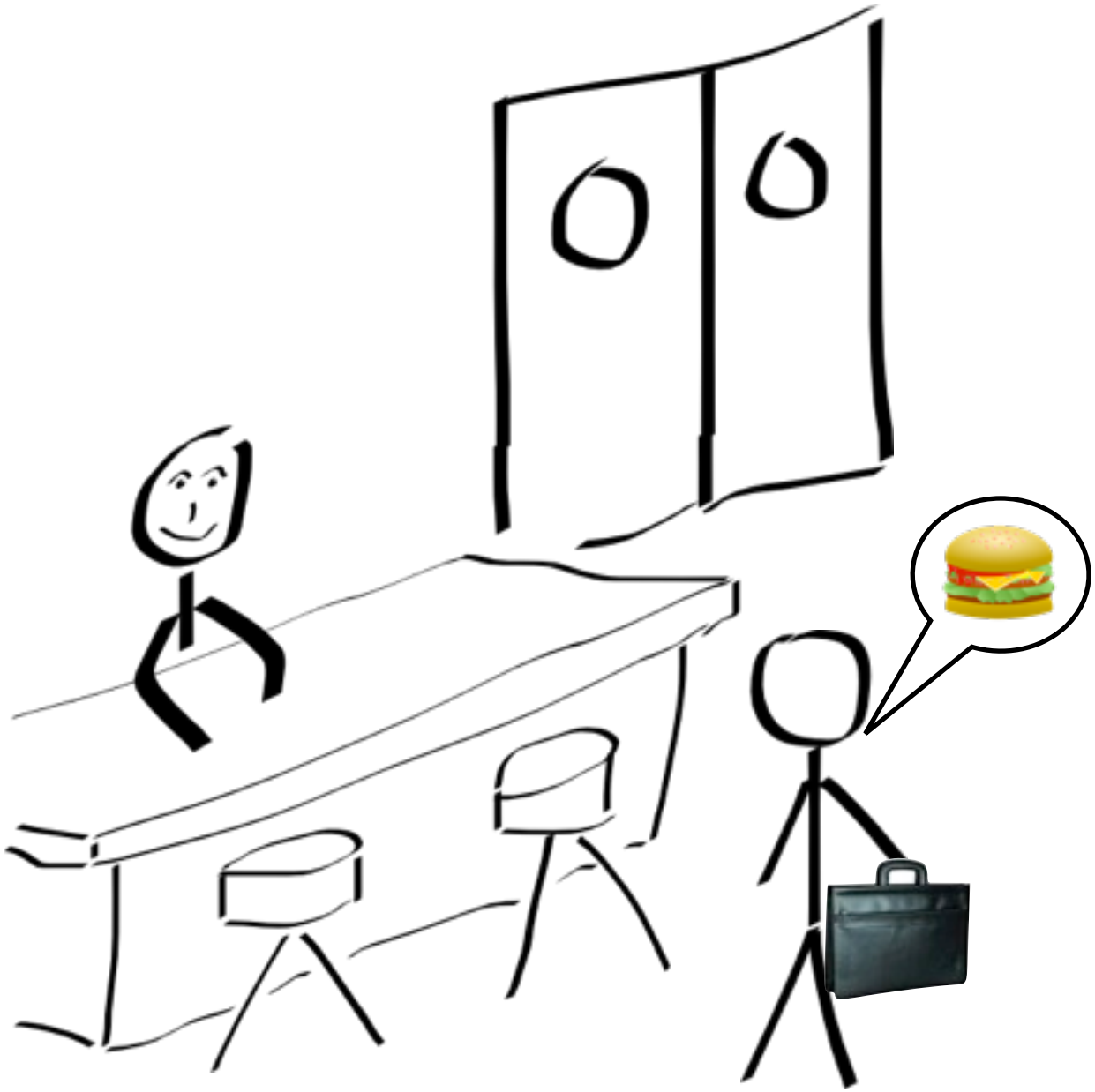
```
# make hamburgers till everyone is fed

while(num_customers>0)

{

        make hamburger

}
```

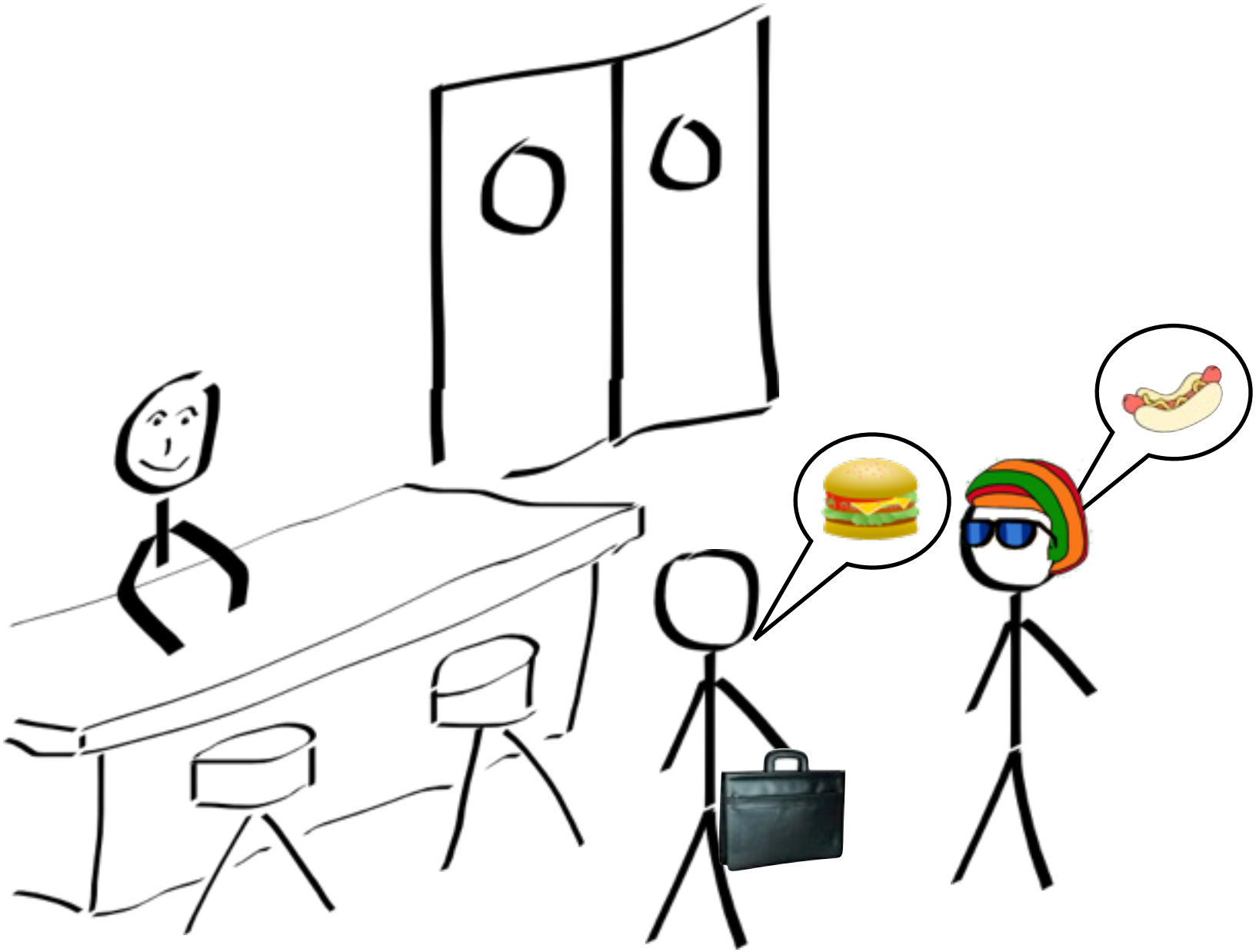
An aside: random numbers

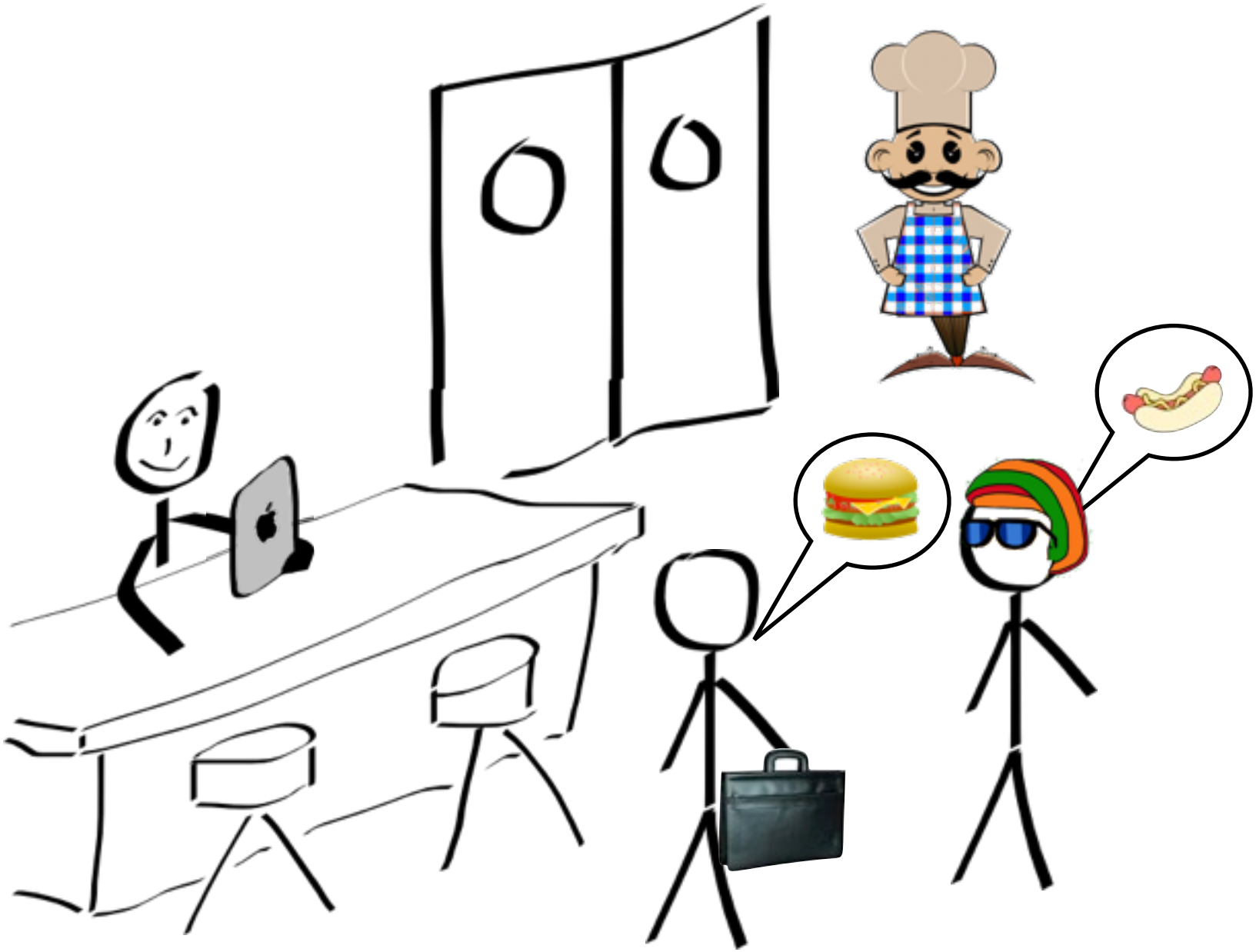
- Draw a random number from a uniform distribution
  - `runif(`*`n`*`, max=0, min=1)`
  - e.g., `runif(1, max=0, min=10)`
- Other distributions:
  - `rnorm(`*`n`*`, mean = 0, sd = 1)`
  - `rbinom(`*`n, size, prob`*`), rexp(`*`n`*`, rate=1),...`

Exercise 9



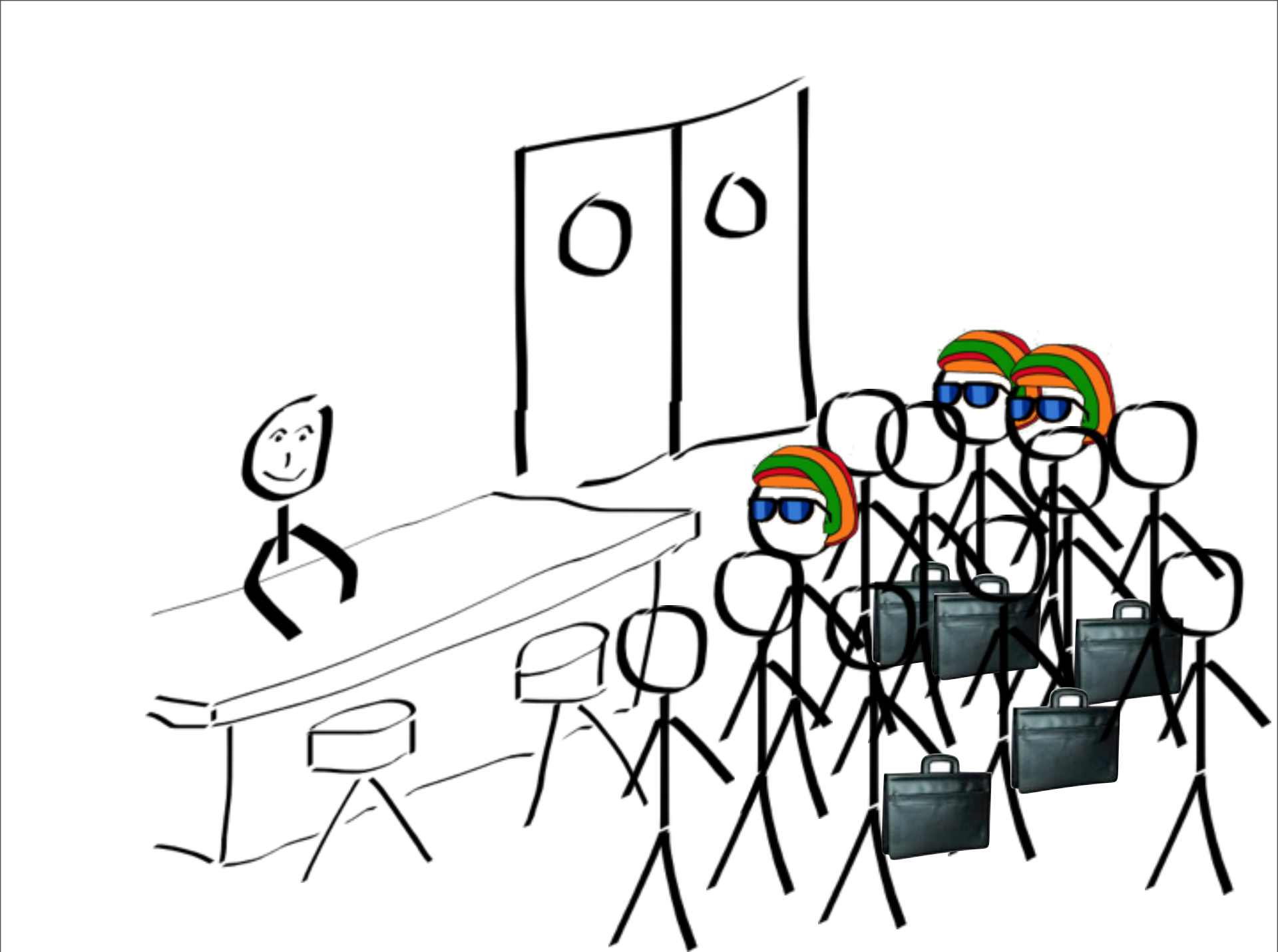






Control structures: `if-else` statements

```
# if possible, decide what to feed customer based on
stereotype

if(businessperson==TRUE)
{
        make hamburger
} else if(hippie==TRUE)
{
        make tofu dog
} else
{
        ask what they want
}
```



Control structures: nesting

```
# make hamburgers till everyone is fed
 while(num_customers>0)
 {
        # if possible, decide what to feed customer
        based on stereotype
        if(businessperson==TRUE)
        {
           make hamburger
        } else if(hippie==TRUE)
        {
           make tofu dog
        } else
        {
           ask what they want
        }
 }
```

Control structures: nesting

```
while(num_customers>0){

   if(businessperson==TRUE){

make hamburger} else if(hippie==TRUE){

make tofu dog} else{

ask what they want}}
```

Proper indenting is important!

Logical operators for control structures

greater than	>
less than	<
greater than or equal	>=
less than or equal	<=
equal	==
AND	&&
OR	||
NOT	!

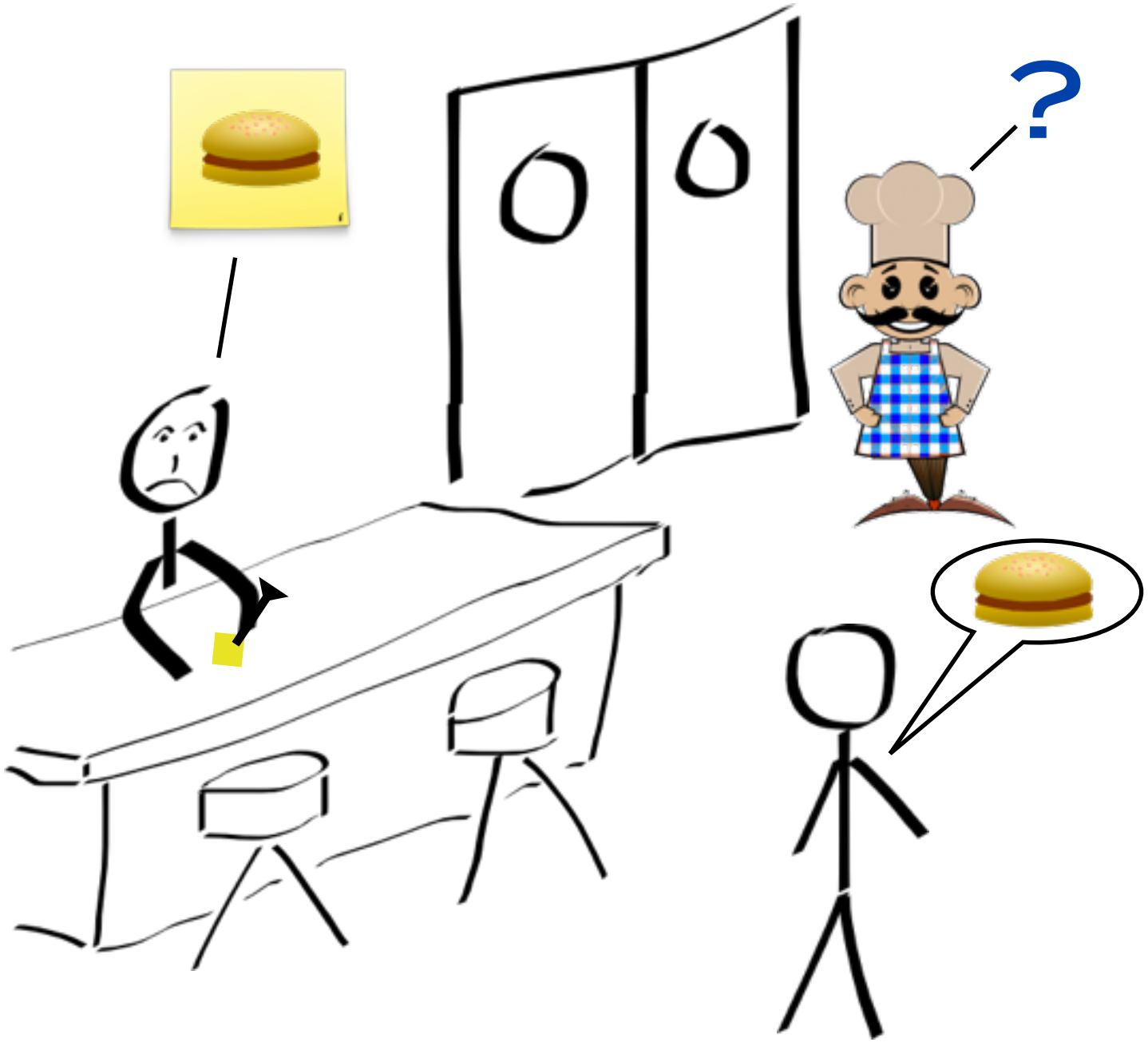
Logical operators: examples

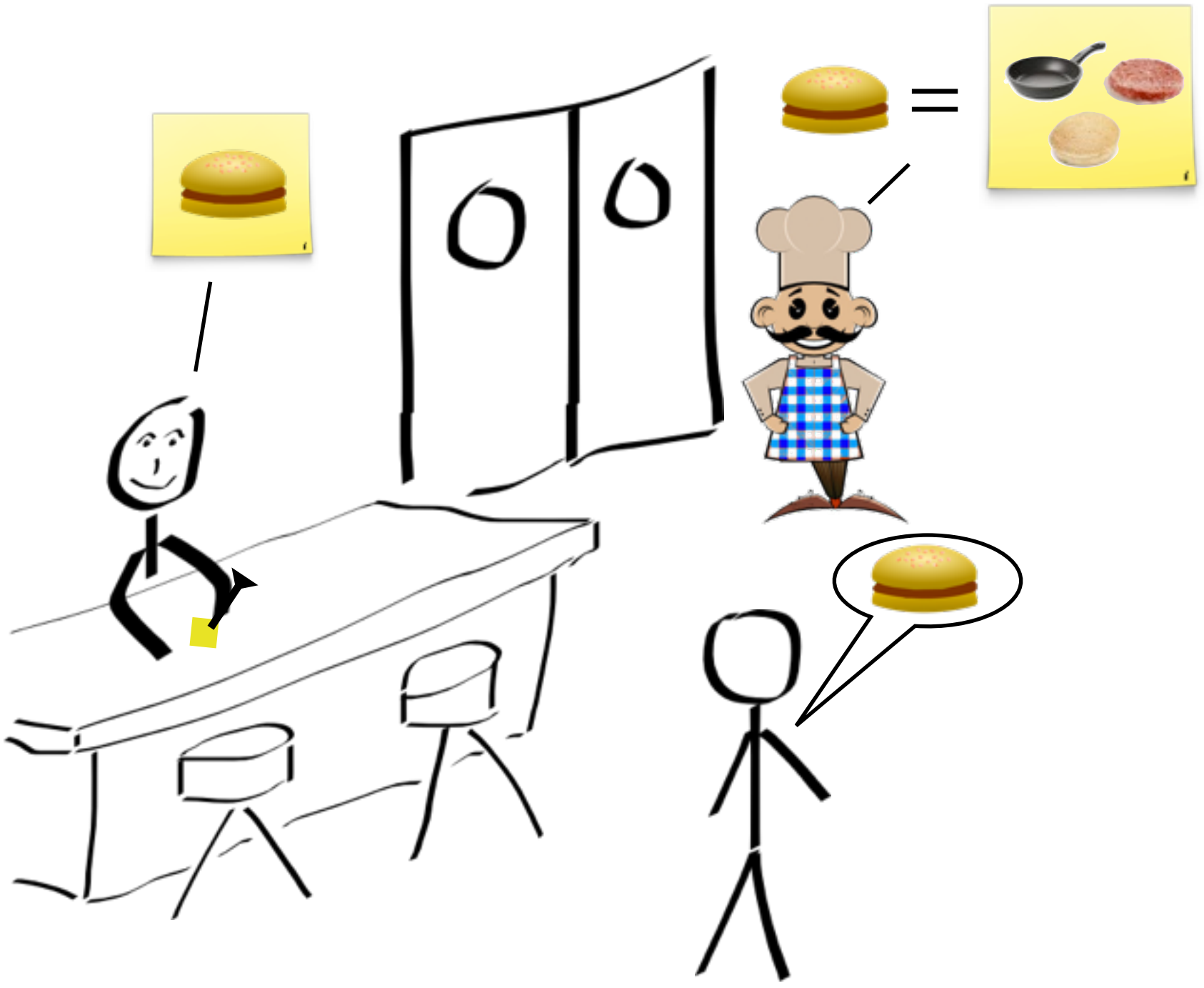
```
# feed Yvon Chouinard

if(businessperson && hippie)

{

        make veggie burger

}
```

Logical operators: examples

```
# don't sell beer to minors
if((age>35) || (age>=21 && hasValidID))
{
        serve alcohol
}
```

Exercise 10





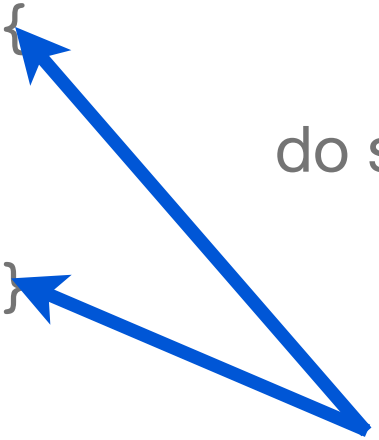
Functions

- defining a function

```
make_hamburger<-function(number_of_patties)
{
    do stuff required to make hamburger
}
```

function name

input parameters



Important!
Braces mark beginning and end of function

Functions

- defining a function

```
make_hamburger<-function(number_of_patties)
{
        do stuff required to make hamburger
}
```

- calling a function

```
make_hamburger(2)
```

Functions

- general form

```
function_name<-function(p1, p2, p3, ...)
{
        do stuff required to compute return_value
        return(return_value)
}
```

Functions: example

```
distance <- function(x1, y1, x2, y2)
{
        # calculate Euclidean distance
        # between two points
        output <- ((x2-x1)^2+(y2-y1)^2)^0.5
        return(output)
 }
 distance(1, 2, 3, 4)
```

Exercise 11

Putting it all together: an example

`tree_dispersal.R`

R resources

Doug Jackson and Dave Allen
31-Aug-2011

Packages

- This is a major strength of R!
- Where can I find information about available packages?
  - http://cran.r-project.org/
    - click on “Packages” link on left hand side of page

Packages

- This is a major strength or R!
- Where can I find information about available packages?
  - http://cran.r-project.org/
    - click on “Packages” link on left hand side of page
- How can I install a package?
  - `install.packages(pkgs='package_name')`
  - Package Installer

Packages

- What packages are on my machine?
  - `library()`

Packages

- What packages are on my machine?
  - `library()`
- How do I load a package so I can use it?
  - `library(`*`package_name`*`)`
  - `require(`*`package_name`*`)` inside functions

Packages

- What packages are on my machine?
  - `library()`
- How do I load a package so I can use it?
  - `library(`*`package_name`*`)` (Note: no quotes!)
  - `require(`*`package_name`*`)` inside functions
- Which packages are already loaded?
  - `search()`

Important things we didn't cover

- Vectorization
  - a way of manipulating elements in a matrix without loops
  - much faster than explicit loops
- Differential equations
  - `deSolve` package

Websites

- The R Project for Statistical Computing
  - http://www.r-project.org/
- R Graph Gallery
  - http://addictedtor.free.fr/graphiques/

Websites

- Teaching modules for R in ecology available through the open-access journal The Plant Health Instructor

  - http://www.apsnet.org/edcenter/advanced/topics/EcologyAndEpidemiologyInR/Pages/default.aspx

- Course website for “Statistical models in Ecology using R”

  - http://magister.ucdavis.edu/~jwwhite/EMD/EMD.html

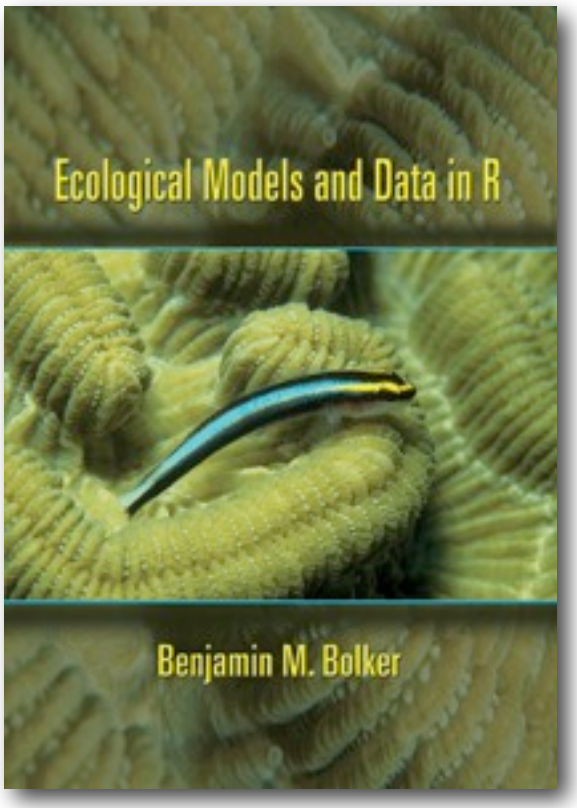
- And of course, if you’re in trouble, Google it!

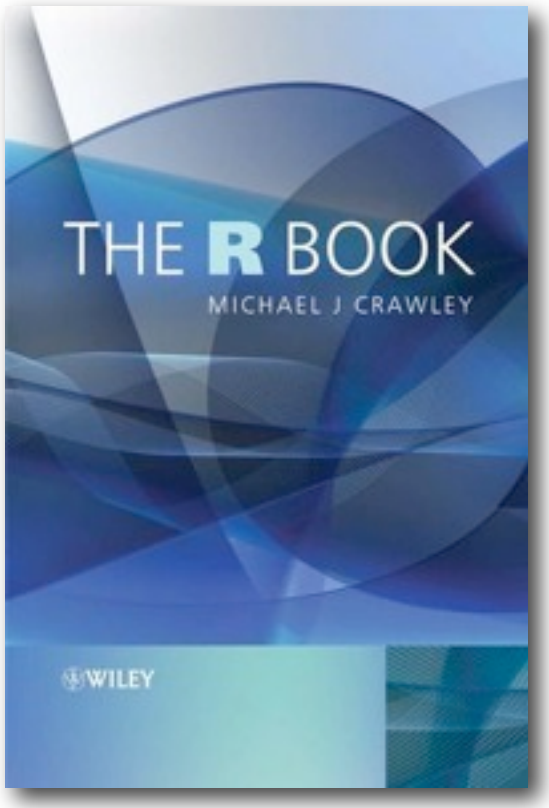
  - (or Bing it, Yahoo! it, etc.)

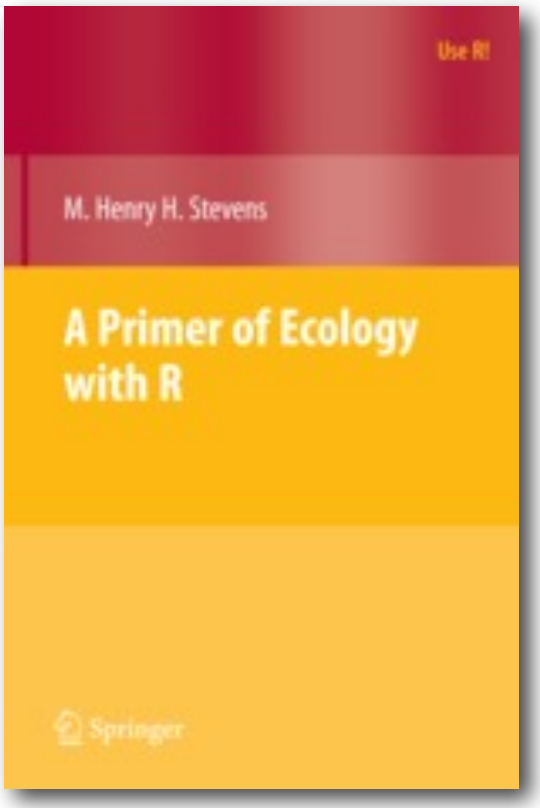
EEB Technical Anger Management ListServe: eeb-tech

- get help, share tips, and generally discuss programming, math, and statistics
- eeb-tech@googlegroups.com
- Join here: http://groups.google.com/group/eeb-tech

Books


Ecological Models and Data in R
Benjamin M. Bolker


THE R BOOK
MICHAEL J CRAWLEY
WILEY


Use R!
M. Henry H. Stevens
A Primer of Ecology with R
Springer

Capstone Exercise