Today

- Still programming focused!
- C for repetition structure (cont)
- C arrays
- Other structures in C & Python
- Functions

C for and arrays

See code

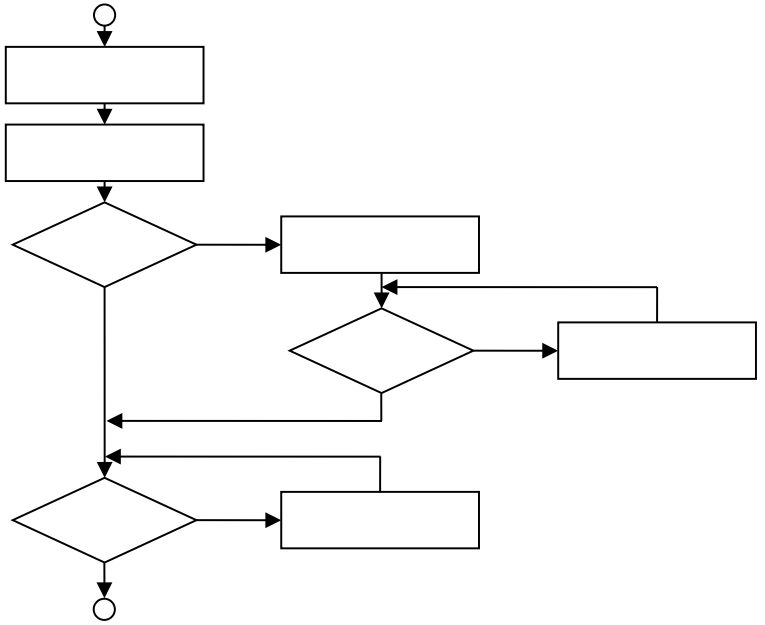
Structured programming

- Sequence structure
- Selection structure (conditional, branches)
- Repetition structure (iteration, loops)

Combining control structures

Stacking
- one after another

Nesting
- one inside another



These are all the programming tools you need to solve any (solvable) problem!

Next: additional, powerful programming tools for convenience or to solve specific problems.

R selection structures

if single selection structure

```
if ( condition ) {
    expression
}
```

if-else double selection structure

```
if ( condition ) {
    expression1
} else {
    expression2
}
```

if-else if multiple selection structure

```
if ( condition ) {
    expression1
} else if {
    expression2
} else {
    expression3
}
```

R repetition structures in practice

while sentinel control

```
while ( condition ) {
    expression
}
```

for counter control

```
for ( i in 1:n ) {
    expression
}
```

until sentinel control

```
while ( !condition ) {
    expression
}
```

foreach collection control

```
for ( item in collection ) {
    expression
}
```

do-while sentinel control (e.g. option 4)

```
repeat {
    expression
    if ( !condition ) break
}
```

R

C selection structures

if single selection structure

```
if ( condition ) {
    expression;
}
```

if-else double selection structure

```
if ( condition ) {
    expression1;
} else {
    expression2;
}
```

if-else if multiple selection structure

```
if ( condition ) {
    expression1;
} else if {
    expression2;
} else {
    expression3;
}
```

C repetition structures in practice

while sentinel control

```
while ( condition ) {
    expression;
}
```

until sentinel control

```
while ( !condition ) {
    expression;
}
```

do-while sentinel control

```
do {
    expression;
} while ( condition )
```

do-until sentinel control

```
do {
    expression;
} while ( !condition )
```

for counter control

```
for ( int i=0; i < n; i++ ) {
    expression;
}
```

Python selection structures

if single selection structure

```
if condition:
    expression
```

if-else double selection structure

```
if condition:
    expression1
else:
    expression2
```

if-else if multiple selection structure

```
if condition:
    expression1
elif:
    expression2
else:
    expression3
```

Py

Python repetition structures in practice

while sentinel control

```
while condition:
    expression
```

for counter control

```
for i in range(n):
    expression_with_i
```

until sentinel control

```
while not condition:
    expression
```

for counter control (Pythonic)

```
for _ in range(n):
    expression
```

do-while sentinel control (Pythonic)

```
while True:
    expression
    if not condition:
        break
```

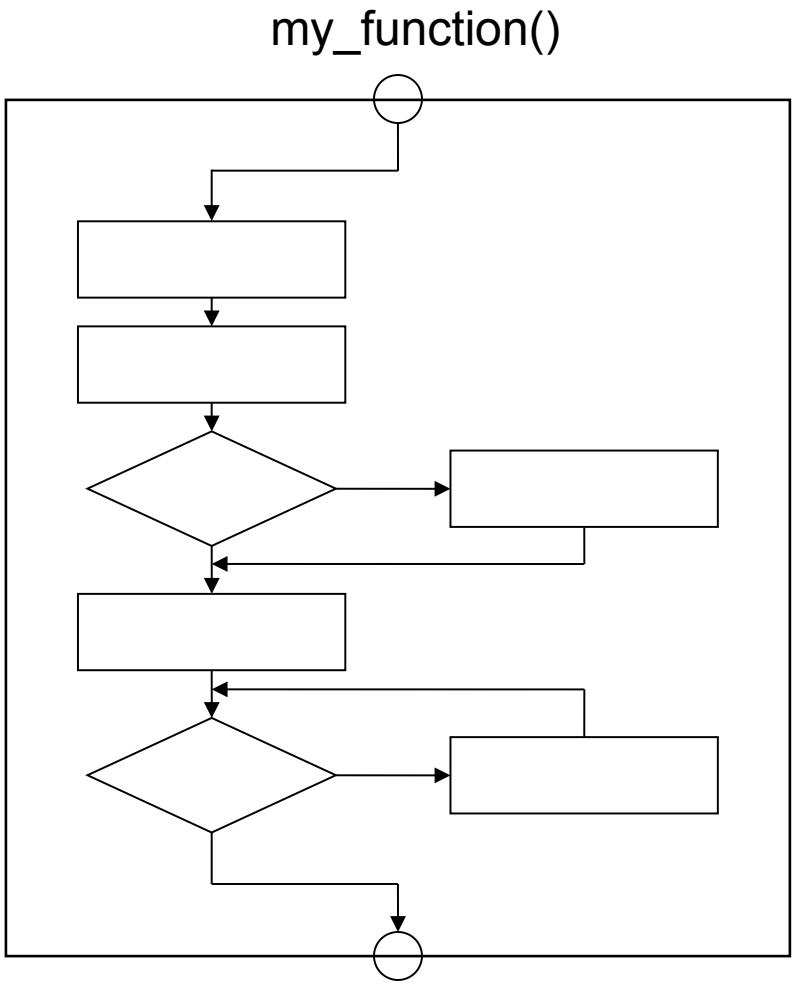
foreach iterable control

```
for item in iterable:
    expression
```

Py

Programming: functions

- A function encapsulates an algorithm
- Functions break a program down into modules
- Modularized programs are easier to write, debug, maintain, and modify
- Functions make algorithms easier to reuse



Making a function in R

- ?"function" – only the bare bones

```
function_name <- function(arguments) {
    expression
    return(object)
}
```

Class style

explicit return

indent (4 spaces)

closing brace aligns with first letter of function name

Making a function in R

- ?"function" – only the bare bones

```
function_name <- function(arguments) {
    expression
    return(object)
}

diff_two_nums <- function(x, z) {
    y <- x - z
    return(y)
}
```

Making a function in R

- ?"function" – only the bare bones

```
function_name <- function(arguments) {
    expression
    return(object)
}

diff_two_nums <- function(x, z) {
    y <- x - z
    return(y)
}
```

Objects listed in the arguments or defined in the function can only be seen inside the function. These are called local variables. Concept: scope.

Scope

- See examples in functions.R
- Good programming practice: avoid global variables
  - Define local variables by including in argument list or initializing within the function
  - Global variables make programs harder to maintain and debug

Make a function

```
function_name <- (arguments) {
    expression
    return(object)
}
```

Break out the movement of the squirrel to a neighboring square

Make a function

```
function_name <- (arguments) {
    expression
    return(object)
}
```

Exercise:
Make a function to calculate the linear model given the model parameters and a vector of x data. In other words, turn the following into a function:

```
y <- b_0 + b_1 * x
```