
Learning To
PROGRAM

So what is a

# PROGRAM

# Most People Think a Program Is

# CODE

```
C040: C0 4C 2B C0 AD 00 DC C9 8D   C210: 69 00 8D FD C1 CA D0 DE 41
C048: 6F D0 E0 AD 83 C1 C9 05 2B   C218: 60 AD FD C8 D0 1E EE F9 C4
C050: F0 D9 EE 83 C1 A9 01 8D 87   C220: C8 AD F9 C8 C9 08 D0 14 10
C058: FD C8 AE 83 C1 BD 69 C1 FB   C228: A9 00 8D F9 C8 EE FF 07 18
C060: AA A9 BA 9D 00 D0 A9 86 0E   C230: AD FF 07 C9 E3 D0 05 A9 12
C068: 9D 01 D0 A9 E3 8D FF 07 F9   C238: E0 8D FF 07 AD 19 D0 29 6E
C070: AE 83 C1 AD 15 D0 5D 6F C4   C240: 01 F0 42 8D 19 D0 20 2C 38
C078: C1 8D 15 D0 A9 01 8D FC E2   C248: C1 CE 16 D0 AD 16 D0 C9 1E
C080: C8 9D 75 C1 4C 2B C0 A2 F8   C250: D0 D0 2F EE F9 C1 AD F9 73
C088: 00 BD CF C4 9D 83 06 A9 AB   C258: C1 C9 D8 D0 1A 20 AB C1 35
C090: 01 9D 83 DA E8 E0 21 D0 49   C260: 20 88 C2 AD FE C8 C9 0C 17
C098: F0 60 60 EE FA C8 AD FA A5   C268: 90 03 EE 82 C1 A9 FF 8D 66
C0A0: C8 C9 02 D0 F5 A9 00 8D 33   C270: 83 C1 A9 00 8D F9 C1 20 C8
C0A8: FA C8 AD FC C8 F0 25 AE A4   C278: E5 C1 20 2C C1 A9 D7 8D 3D
C0B0: 83 C1 BD 69 C1 AA DE 01 69   C280: 16 D0 4C BC FE 4C 31 EA D7
C0B8: D0 FE 00 D0 FE 00 D0 EE 18   C288: A2 00 BD 75 C1 D0 03 20 14
C0C0: FB C8 AD FB C8 C9 06 D0 98   C290: 94 C1 E8 E0 06 D0 F3 A2 1E
```

That's

WRONG

Code is

**HOW**

you make a program

A program is just a set of

**INSTRUCTIONS**

# Could a loom be a

# PROGRAM ?


https://www.youtube.com/watch?v=OlJns3fPItE

Anyone have an

# IKEA BED?

They require **INSTRUCTIONS**

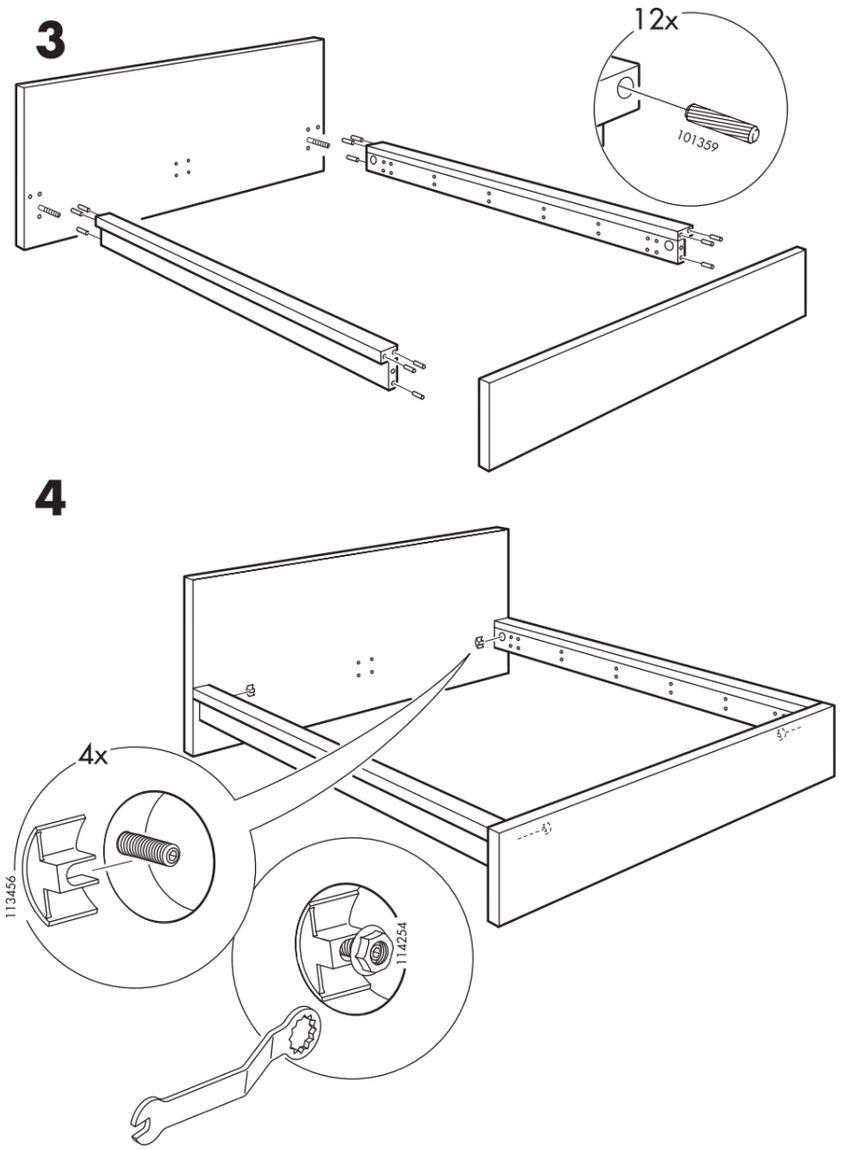
to build

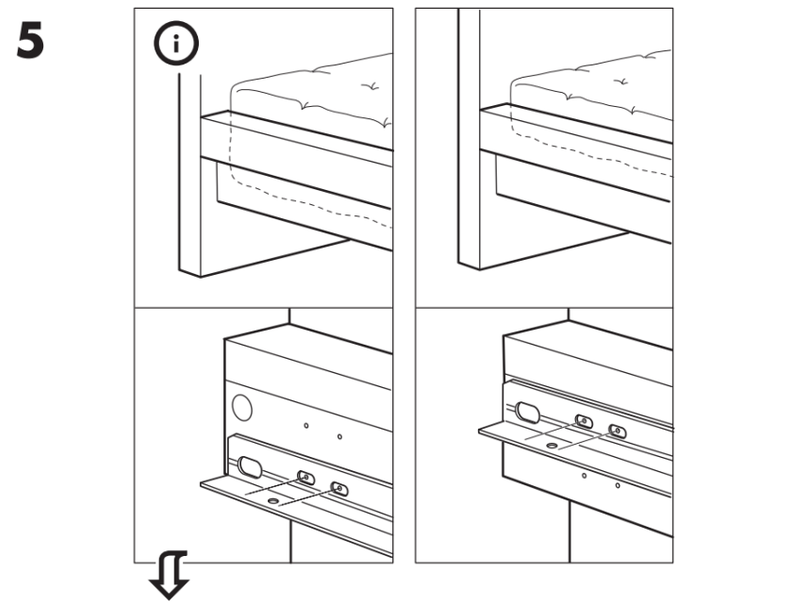
Instruction Manuals are

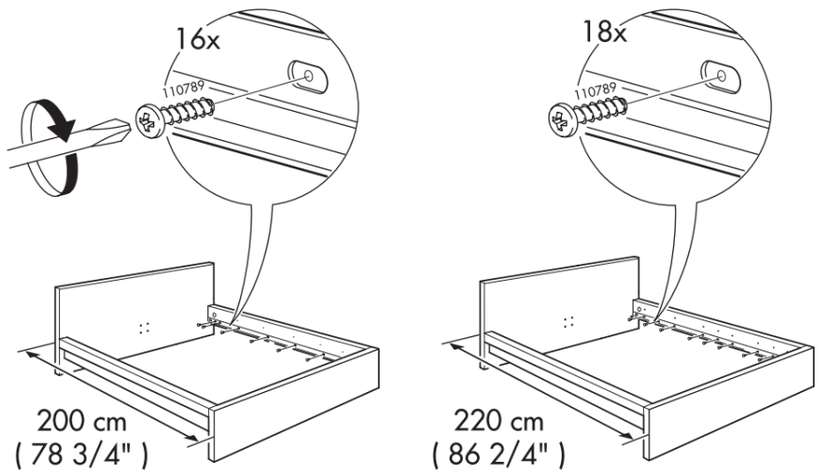
PROGRAMS

# Can You Describe

# STEPS 3-5?






AA-75286-12

# Thinking about programs as

# PSEUDO CODE

# IKEA BED PROGRAM

```
bedBuild
  attach bedSills to headboard and footboard
  tighten bedSills fasteners
  secure bedRails
```

# THE DISCONNECT

You might have noticed how dumb computers appear to be. Why is that?

# EXAMPLE

I say, "I'm going to become the next Miss America".

Human: Laughs (maybe cries?)
Computer: Ok, now what?

# CONTEXT

Computer doesn't know I'm a late-20's male programmer.

It NEVER KNOWS context. The art of thinking like a computer is to assume nothing. It only knows what you tell it.


Who is
HUNGRY?

How about a

# STIR-FRY?

Because recipes are

PROGRAMS

# STIR FRY PROGRAM

```
spices = soySauce + riceVinegar + sugar
oil = 1.5 ounces
brownRice = 1.5 cups
broccoli = 2 cups
shrimp = .5 pounds
```

```
stirFry {
 cook brownRice
 whisk spices
 heat oil in pan
 add broccoli
 cook shrimp
 add spices
 }
```

# STIR FRY PROGRAM

```
spices = soySauce + riceVinegar + sugar
oil = 1.5 ounces
brownRice = 1.5 cups
broccoli = 2 cups
shrimp = .5 pounds

stirFry(spices, oil, brownRice, broccoli, shrimp)
 cook(brownRice, .25)
 whisk spices
 heat oil in pan
 add broccoli
 cook shrimp
 add spices
 }
```

Recipe verbs are functions you can use with the ingredient variables

The ingredients act as variables being passed into the stirFry function.

# VARIABLES

Reusable object in a program

# FUNCTIONS

Modular container for an instruction set (more on this tomorrow)

# JAVASCRIPT

## Have Assignment 1 Open

# WHAT IS JS?

-Most basic way to understand it:

Frontend: The dynamic parts of a webpage

Backend:  Everything (if it is using JS at all)

# WHAT CAN JS DO?

1) Respond to user actions (events)

2) Change HTML Content / CSS Styles

3) Loading/sending data to and from the server

# WHAT MAKES JS SPECIAL?

-All programming languages require code to be compiled before viewing

-Javascript is unique because the compiler can be your browser, so you can see your work immediately

# HOW IT LOOKS

```
var firstFlight = null;

function kittyHawk(firstFlight) {
   if (firstFlight >= 1903) {
      console.log("We're airborne");
   }
   else {
      console.log("Let us dream...");
   }
}
```

# DATA TYPES

⇒ Numbers

⇒ Strings

⇒ Booleans

⇒ Null/Undefined

⇒ Functions

⇒ Objects (for later)

# NUMBERS

-You've probably seen these before:
`0, 1, 2, 3, 4, 5, 6, 7, 8, 9`

-Used mathematically throughout JS code, so normal math rules apply

-Paired with operators:
`+, -, *, /, %`

# NUMBERS

```
10*10
  ⇒ 100

8-4
  ⇒ 4

49/7
  ⇒ 7
```

# MODULUS OPERATOR

- The “%” operator is a special one.
- It will give you back the remainder of any division statement.
- For example 7 % 3 will return 1
- Because 7/3 = 2, but there is still 1 remaining.
- Other examples:
  - 13 % 4 => 1
  - 28 % 5 => 3
  - 14 % 6 => 2

# What would…

- 10 % 5 return?
- 19 % 4 return?
- 22 % 2 return?
- 7 % 2 return?
- 18 % 2 return?
- 9 % 2 return?
- Any even number % 2?
- Any odd number % 2?

# STRINGS

-String means text, that's it:

```
"It is a beautiful evening";
"Is it really Monday?";
"I am feeling good today!";
```

-With JS, you can merge strings w/ "+" operator, ie:

```
"You want" + "to go" + "eat
on 14th?"
```

# BOOLEAN

-True / false variables in JS, don't over think it

```
var Chinatown = Boolean("Jack Nicholson");
⇒ True
```

```
var apocalypseNow = Boolean("1979");
⇒ True


var smashingPumpkinsEP = Boolean("0");
⇒ False
```

# NULL / UNDEFINED

-Values that denote nothingness in Javascript:

```
var Lysine;
console.log(Lysine);
⇒ undefined
```

```
console.log(9 * null);
⇒ 0
```

```
console.log("five" * 5);
⇒ NaN
```

# MOST BASIC OUTPUT

```
// Writes to console
console.log("Houston, do you
copy?");
```

# CHECK OUT ASSIGNMENT 1

# CODEALONG

# MATH TOOL

Have Assignment 1 Open

# MATH TOOL

- Javascript also has a handy tool, the Math tool
- The math tool offers handy tools for doing math-related things.  For example:
- Math.pi will return Pi to a high precision
- Math.round(0.4) will return 0
- Math.round(0.7) will return 1
- Math.pow(4, 2) will return 16
- Math.ceil(4.2) will return 5
- Math.floor(4.2) will return 4

# MATH TOOL

- To be honest, the number of Math functions are in the dozens and would be far too long to list.  There are a few key ones to remember:
- Math.ceil(num)
- Math.floor(num)
- Math.random() will return a random number between 0 and 1.

# MATH.RANDOM()

- Math.random() can be used to get a random number between 0 and any number by doing:
- Math.random() * num
- For example Math.random() * 9 might return 6.66729
- Or 2.94818819
- Or 0.0000172
- How might we combine this method with some of the other Math methods to get a whole number between 0 and any number?

# Variables

# VARIABLES

-Data storage in a program
-Declare all variables with `var` keyword
-'Call' variable just by using it's name somewhere in your program

```
var flyingMonkeys = 5;
var tiredGoats = 7

flyingMonkeys + tiredGoats
⇒ 12
```

# VARIABLE NAMING

-Names should be easily understood
-No spaces allowed
-You can use "-" or "_" but don't
-Use camelCase - itWorksLikeThis

```
var howManyCoasters = 18;
var midWeek = "Wednesday";
```

# VARIABLE REASSIGNMENT

```
var x = 18;
console.log(x);
⇒ 18
```

```
x * 2;
console.log(x);
⇒ 18
```

# VARIABLE REASSIGNMENT

```
var x = 18;
console.log(x * 2);
⇒ 36
```

```
var x = 18 * 2;
console.log(x);
⇒ 36
```

# Conditionals

# COMPARISON OPERATORS

Allow you to compare instead of define values

```
>    (greater than)
>=   (greater than or equal to)
<    (less than)
<=   (less than or equal to)
==   (equality)
===  (strict equality)
!==  (inequality)
```

# COMPARISON OPERATORS

- Comparison operators will always return true or false, depending on if the condition is met.
- For example:
  - `4 === 4  => true`
  - `5 > 3    => true`
  - `5 <= 3   => false`
  - `7 >= 7   => true`
  - `7 !== 7  => false`
- And that's just for numbers, we can also use these for strings and other data types

# Comparison Operators

- For example:
  - `'cat' === 'cat'  => true`
  - `5 === 'kitty'    => false`
  - `true !== true    => false`
  - `false === true   => false`
- There are loads of strange rules that can be useful, but rarely:
- `'cat' < 'dog'  => true`
- `'dog' > 8      => false`

# LOGICAL OPERATORS

- There is another set of operators available to us, ***logical operators.***
- Logical operators include:
  - AND: &&
  - OR: ||
- For an ‘and’ statement, both sides of it must evaluate to “True” to return true
- `true && false  => false`
- `true && true  => true`

# THE '&&' OPERATOR

- You can chain it with other operators:
- `(5 > 3) && (6 < 9)  => true`
- `(8 <= 9) && ('cat' === 'cat')  => true`

- This can be a bit challenging. Using parentheses is a good way to separate your logic and prevent unexpected errors.
- Things in parentheses will evaluate before things outside of parentheses

# THE "||" OPERATOR

- The or operator, '||' needs both side to evaluate to 'true' for the statement to evaluate to 'true'.
- `false || false  => false`
- `false || true  => true`
- `true || false  => true`
- `true || true  => true`

# THE "||" OPERATOR

- You can chain it with other operators:
- `(5 > 3) || (6 < 9)  => true`
- `(false) || ('cat' === 'cat')  => true`

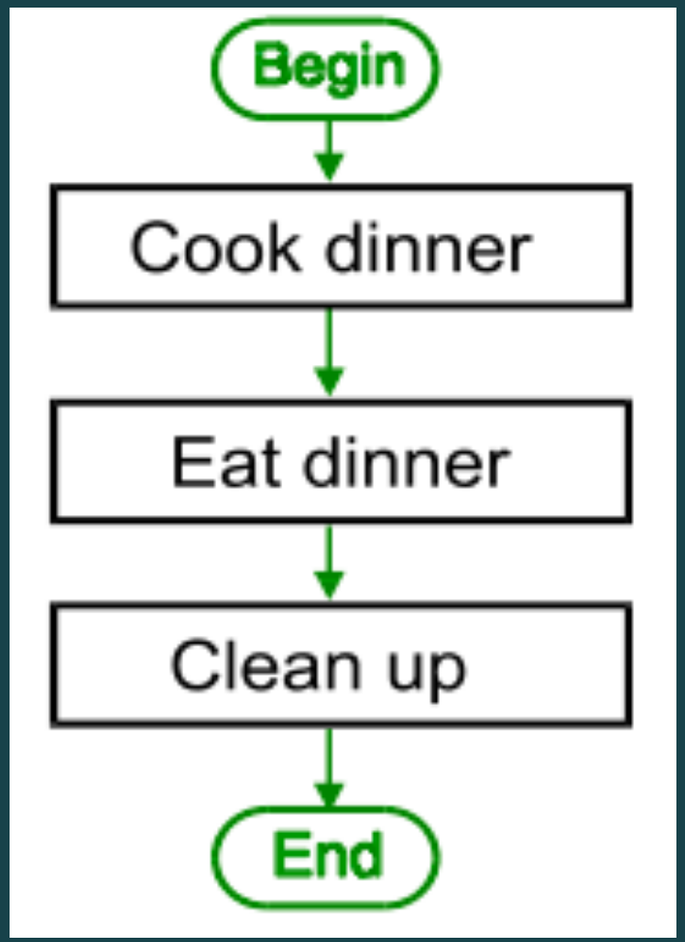
- The 'or' operator is more forgiving

# CONDITIONALS

Conditional statements enable us to essentially decide which blocks of code to execute and which to skip, based on the results of tests that we run.

# CONDITIONALS

- Sequential control flow:
  - Programs execute line by line, from top to bottom:



# CONDITIONALS

- Programs can flow in other ways:

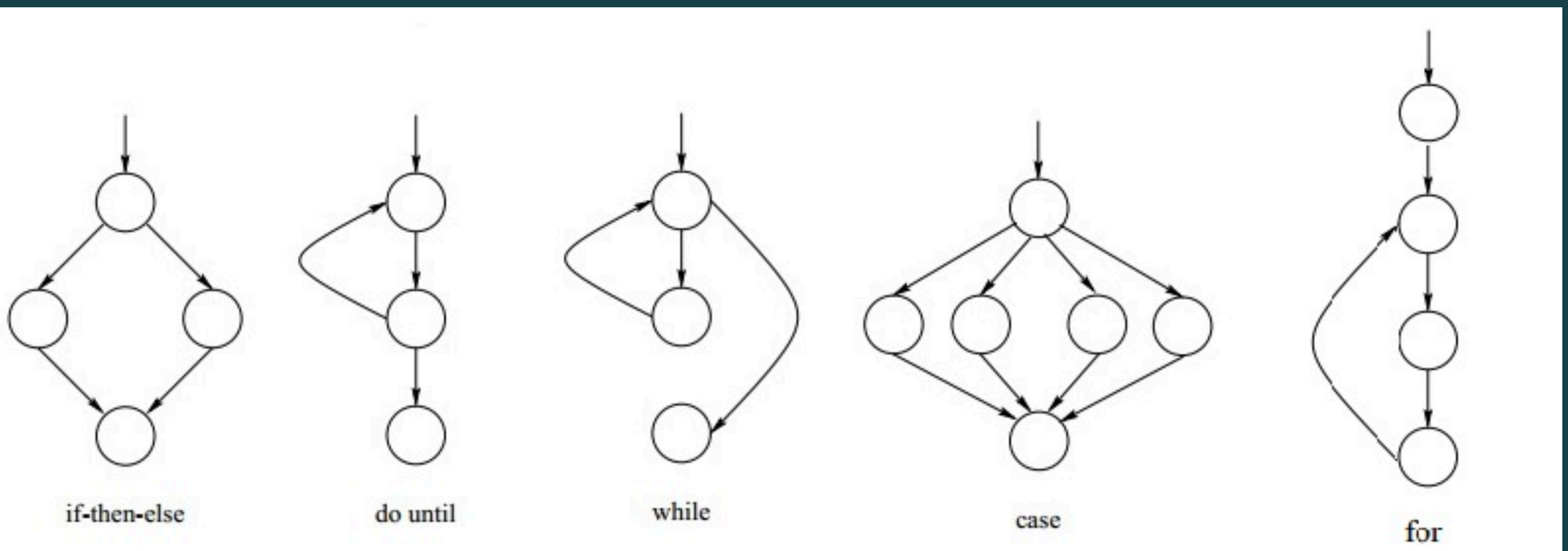


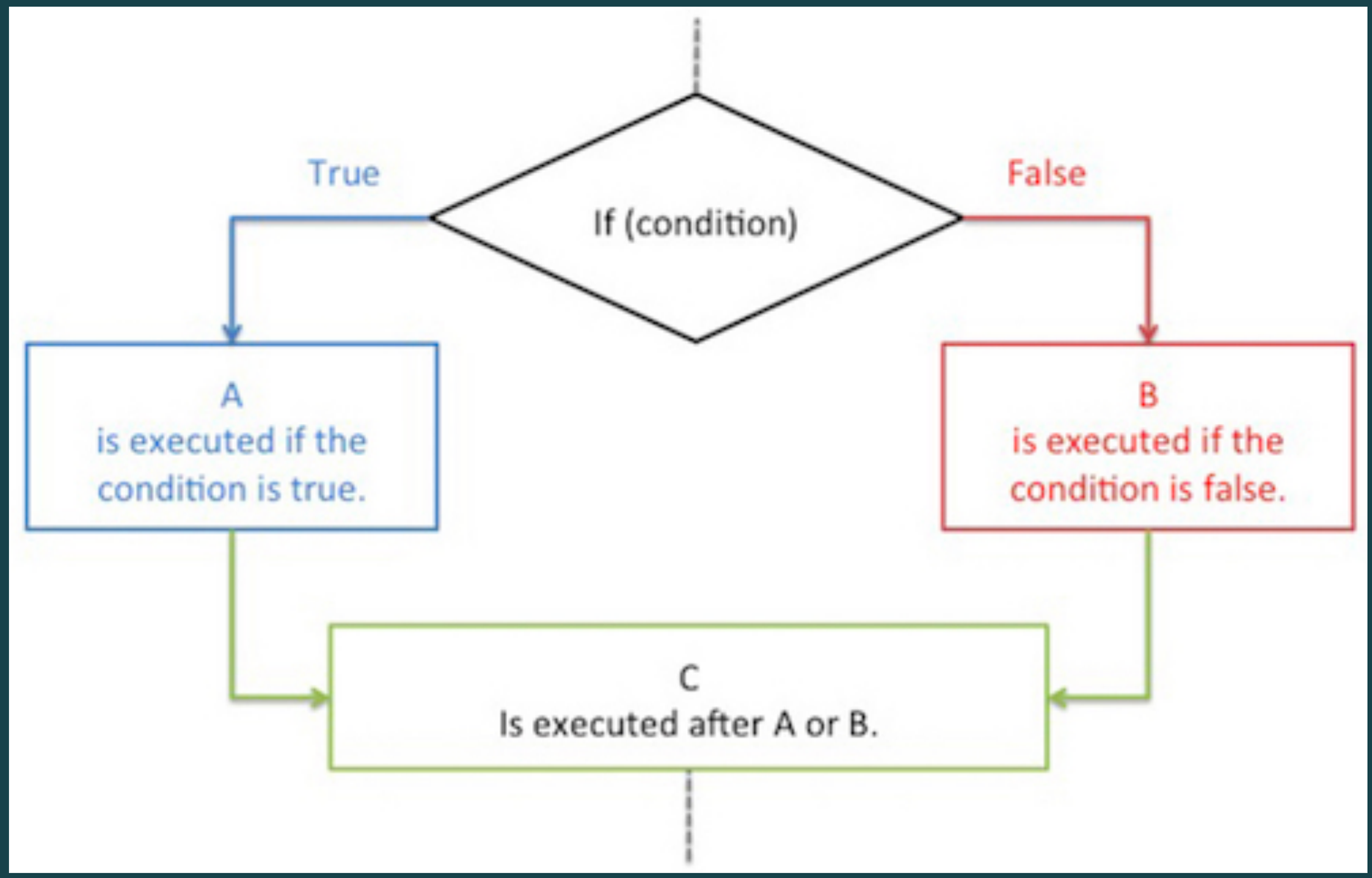

Figure 1: Flow graph representation.

# CONDITIONALS

- Conditonal Flow:
- ⇒ AKA if/else statements
- ⇒ Alters flow when a decision must be made
- ⇒ Choice is based on variables in the program

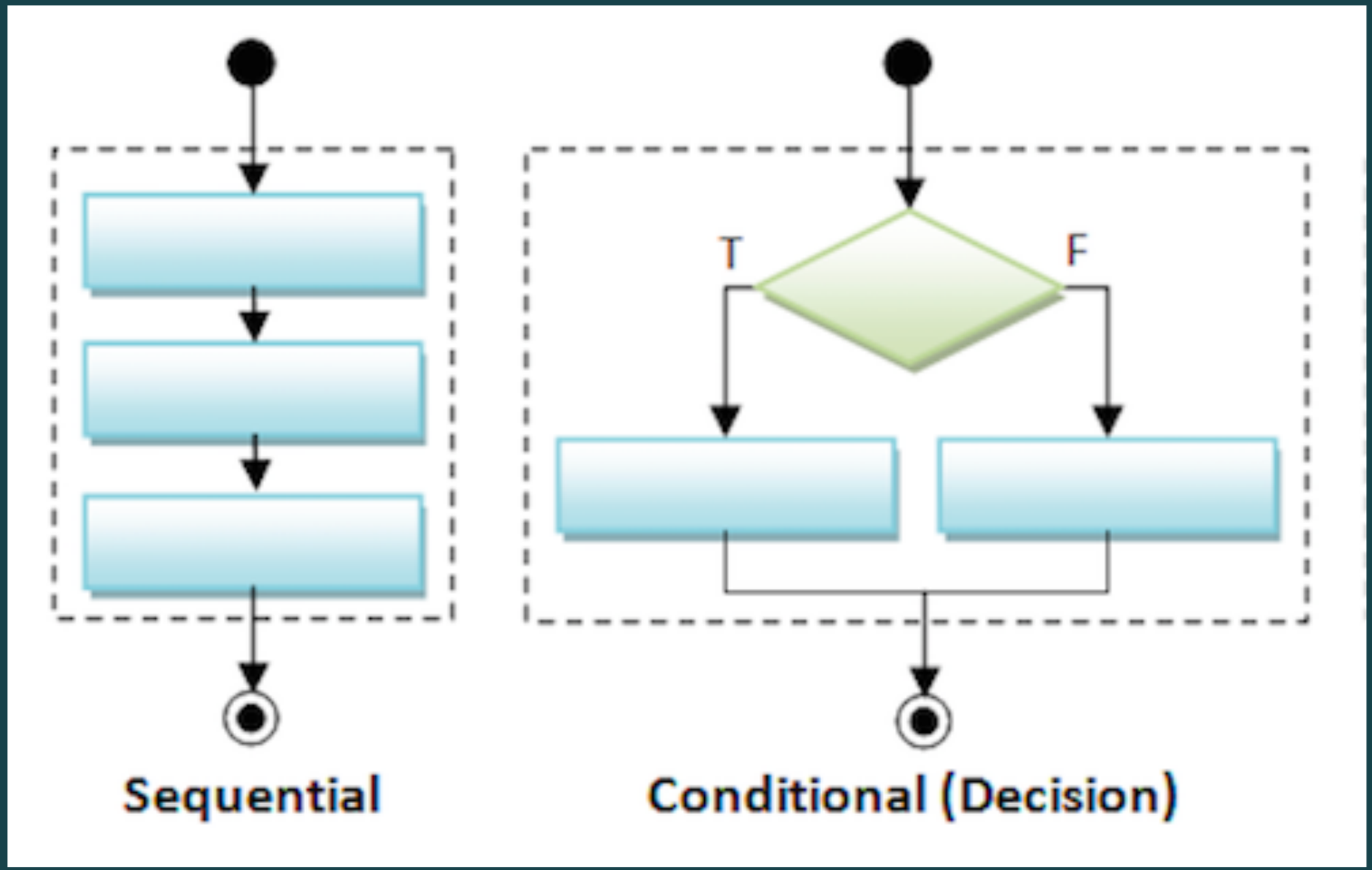
# CONDITIONALS

- If/Else Flow:



# CONDITIONALS

- Sequential vs. Conditional Flows:



# CONDITIONAL STATEMENTS

Most simplistic version

```
if (test1 === test2) {
  // anything in here will be run
}
```

# CONDITIONALS

- The most other common conditional is the if/else statement:

```
if (condition) {
  code
} else {
 other code
}
```

# CONDITIONALS

```
if (condition) {
   code
  } else {
   other code
  }
```

- An if/else statement gives you two different paths.
- The ‘condition’ stands for some kind of condition (a comparison operation )
- And ‘code’ stands for some block of code you want to run if that condition evaluates to ‘true’.
- If that condition evaluates to false, then the ‘other code’ will run.

# COMPARISON OPERATORS IN CONDITIONALS

They usually compare integers or boolean variables within a conditional statement

```
var blackJack = 21, daysInAWeek = 7;

if (blackJack >= daysInAWeek) {
  // valid condition
  // function will execute this code
} else {
  // run this code if the condition
  is not met
}
```

# CONDITIONAL STATEMENTS

If / Else If allows for deep conditional nesting. Be careful with this - don't confuse yourself.

```
if (test1 === test2) {
  // run if first condition is true
} else if (test1 >= test2) {
  // run if second condition is true
} else {
  // runs if neither are true
}
```

# REMEMBER

When you use '=' that defines a value. If you try to use '=' in a comparison statement, you will reassign variables!  DON’T DO THIS:

```
var blackJack = 21, daysInAWeek = 7;
if (blackJack = daysInAWeek) {
   // valid condition
   // function will execute this code
}
```

# LET'S DO SOME TOGETHER

What does this return?

```
var vehicle = 'car';

if (vehicle === 'car') {
  console.log('beep beep');
}
```

What does this return?

```
var animal = ‘cat’;

if (animal !== ‘dog’) {
  console.log(‘what is this?’);
} else {
 console.log(‘I want to pet it’);
}
```

What does this return?

```
var entertainment = 'movie';
var year = 1984;

if (entertainment === 'book') {
  console.log('dim the lights!');
} else if (year <= 1984)  {
  console.log('An oldie but goodie');
} else if (entertainment === 'movie'){
  console.log('grab yer popcorn');
} else {
  console.log('what are we doing?');
}
```

What does this return?

```
var animal = ‘cat’;
var age = 12;

if ((animal === ‘cat’) && (age < 2)) {
  console.log(‘dawww’);
} else if ((animal === ‘cat’) && (age >= 12)({
  console.log(‘so nice!’);
} else {
  console.log(‘not sure about this…’);
}
```

# MORE CODEALONG