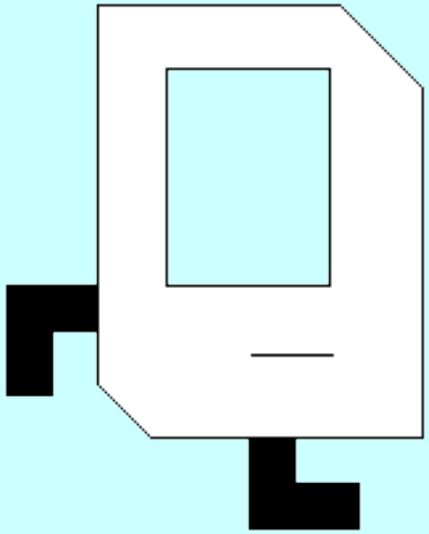
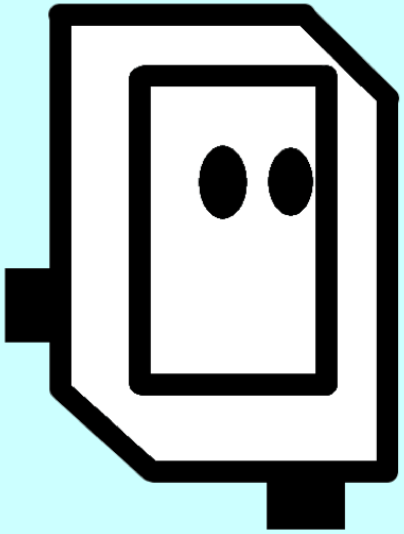
Introductory Programming:
LOGO, Scratch, Karel the Robot, Bit, Pascal, BASIC

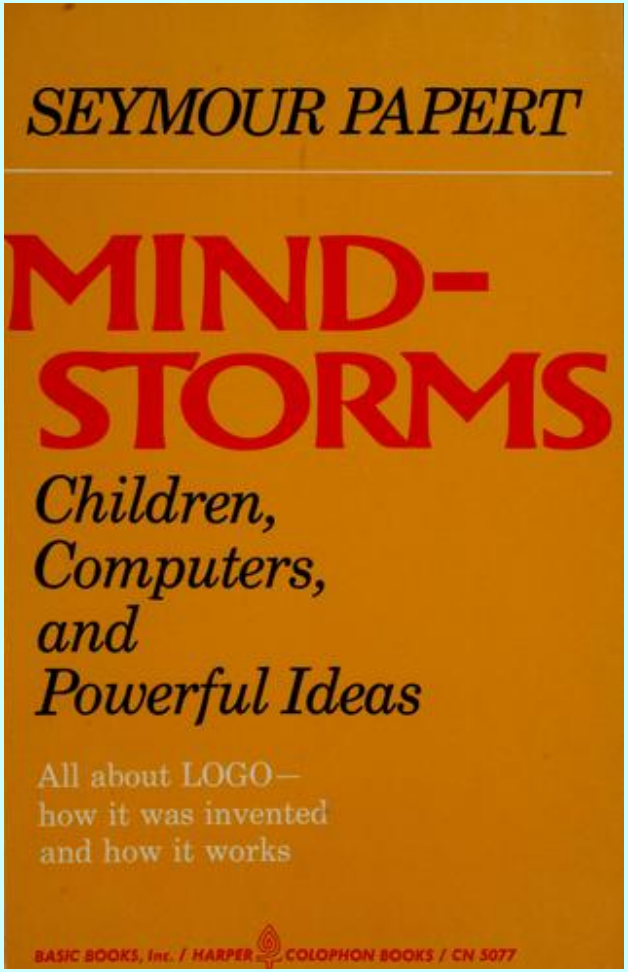




Chris Gregg
Based on Slides from Eric Roberts
CS 208E
Sept 27, 2021

The Project LOGO Turtle

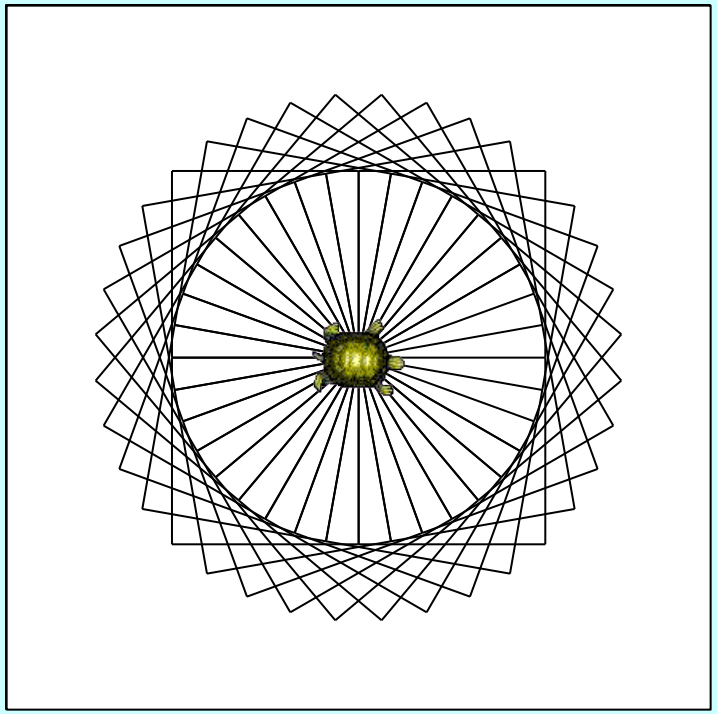
- In the 1960s, the late Seymour Papert and his colleagues at MIT developed the Project LOGO turtle and began using it to teach schoolchildren how to program.
- The LOGO turtle was one of the first examples of a ***microworld***, a simple, self-contained programming environment designed for teaching.
- Papert described his experiences and his theories about education in his book *Mindstorms,* which remains one of the most important books about computer science pedagogy.



Programming the LOGO Turtle

```
to square
  repeat 4
    forward 40
    left 90
  end
end


to flower
   repeat 36
      square
      left 10
   end
end
```



The Logo Turtle in Python

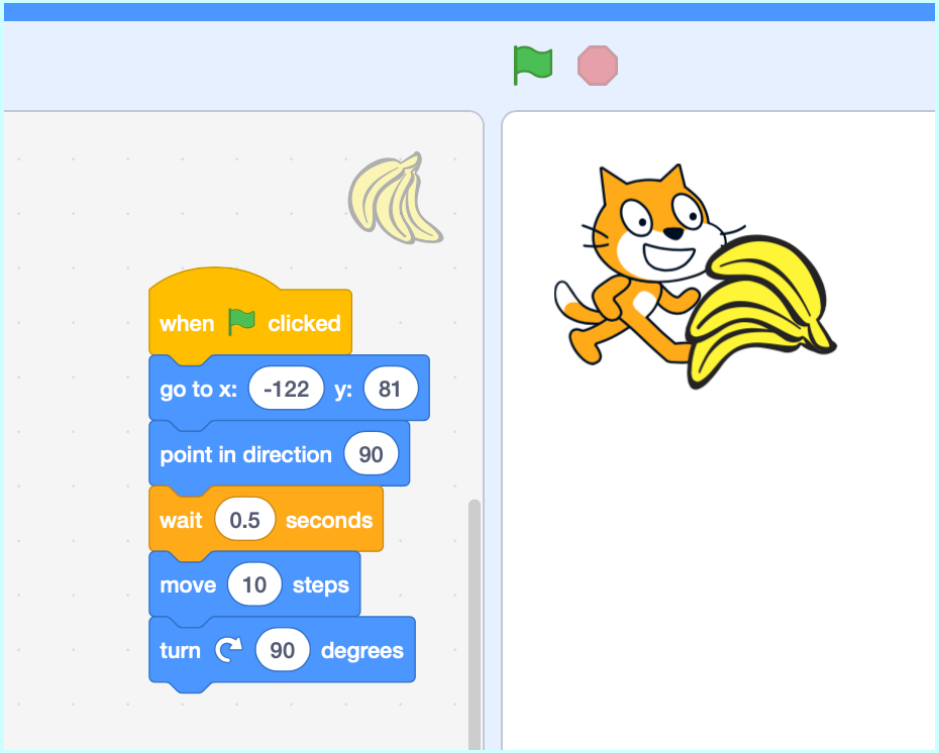
```
% python3
Python 3.9.1 (v3.9.1:1e5d33e9b9, Dec  7 2020, 12:10:52)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from turtle import *
>>> import turtle
>>> def square():
...
KeyboardInterrupt
>>> def square(t):
...   for i in range(4):
...     t.forward(40)
...     t.left(90)
...
>>> def flower(t):
...   for i in range(36):
...     square(t)
...     t.left(10)
...
>>> flower(turtle.Turtle())
```

A Logo Sun in Python

```
% python3
Python 3.9.1 (v3.9.1:1e5d33e9b9, Dec  7 2020, 12:10:52)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from turtle import *
>>> def sun():
...     color('red', 'yellow')
...     begin_fill()
...     while True:
...         forward(200)
...         left(170)
...         if abs(pos()) < 1:
...             break
...     end_fill()
...     done()
...
>>> sun()
```

Scratch

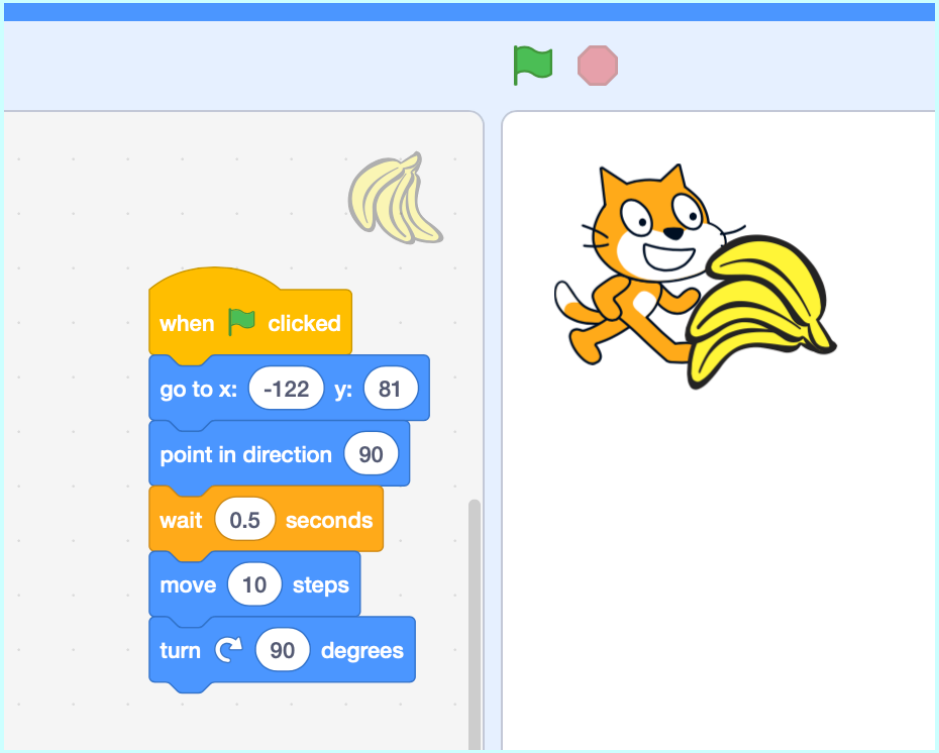
- Scratch is a high-level, "block" based programming language, targeted at children to introduce them to programming.
- It was launched to the public in 2007



- It is a particularly unique language because of its block nature, but also because it is event-driven (e.g., mouse clicks, key presses, timing, etc. determine what happens)
- There are tens of millions of Scratch projects, and Scratch has been listed as one of the top-20 most popular languages

Scratch

- Scratch does have its criticisms:
  - Students who program in Scratch can end up with overly-complicated programs because of the event-driven model.



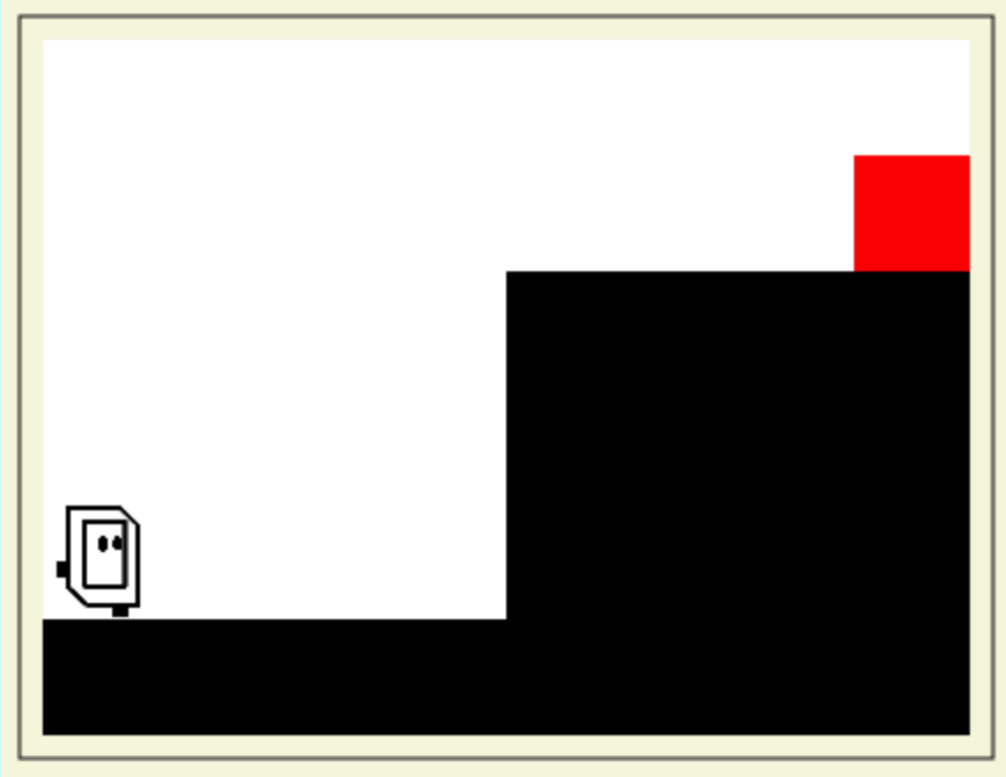
- Students may have a hard time translating from Scratch to text-based languages
- Students "age-out" of scratch (i.e., "this is for kids!") without getting deep enough. Scratch is actually a robust language that has many interesting features, but most students never see those features.

Rich Pattis and Karel the Robot

- Karel the Robot was developed by Rich Pattis in the 1970s when he was a graduate student at Stanford.
- In 1981, Pattis published *Karel the Robot: A Gentle Introduction to the Art of Programming,* which became a best-selling introductory text.
- Pattis chose the name *Karel* in honor of the Czech playwright Karel Čapek, who introduced the word *robot* in his 1921 play *R.U.R.*
- In 2006, Pattis received the annual award for Outstanding Contributions to Computer Science Education given by the ACM professional society.

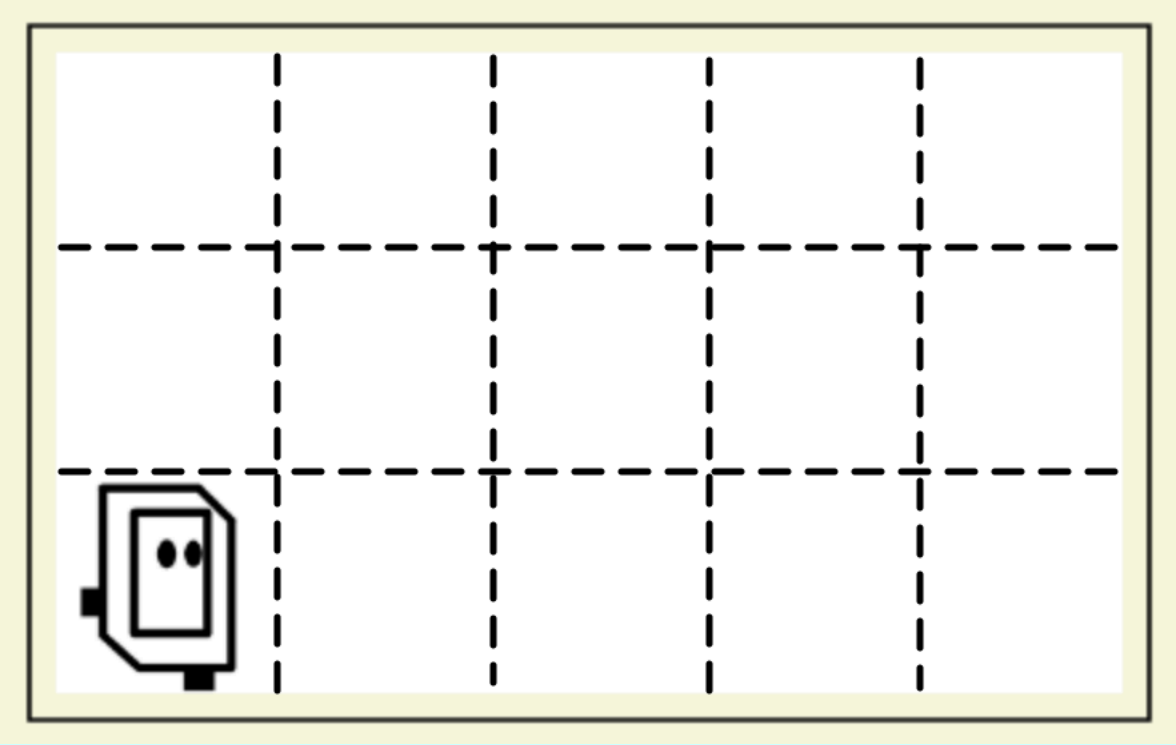
Bit

- Depending on who is teaching CS106A at Stanford, either Karel or *Bit* might be used to introduce programming to students. Karel has been used for many generations at Stanford and other universities.
- Bit is a derivative of Karel (and looks very similar), created by Stanford lecturer Nick Parlante for a revised version of CS106A.



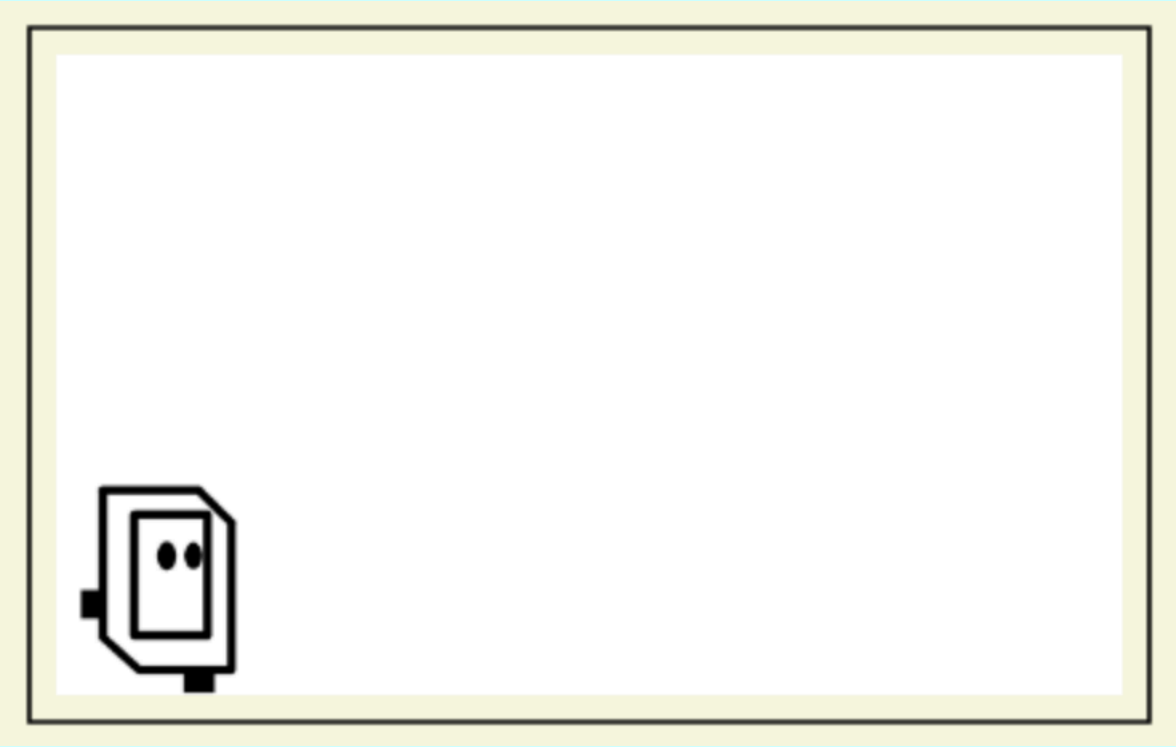
Meet Bit

- Bit lives in a grid world, and can move through the world one square at a time (the dashed grid lines are not shown when using Bit)
- Bit is currently facing to the right, and will move right if given a *move* command.



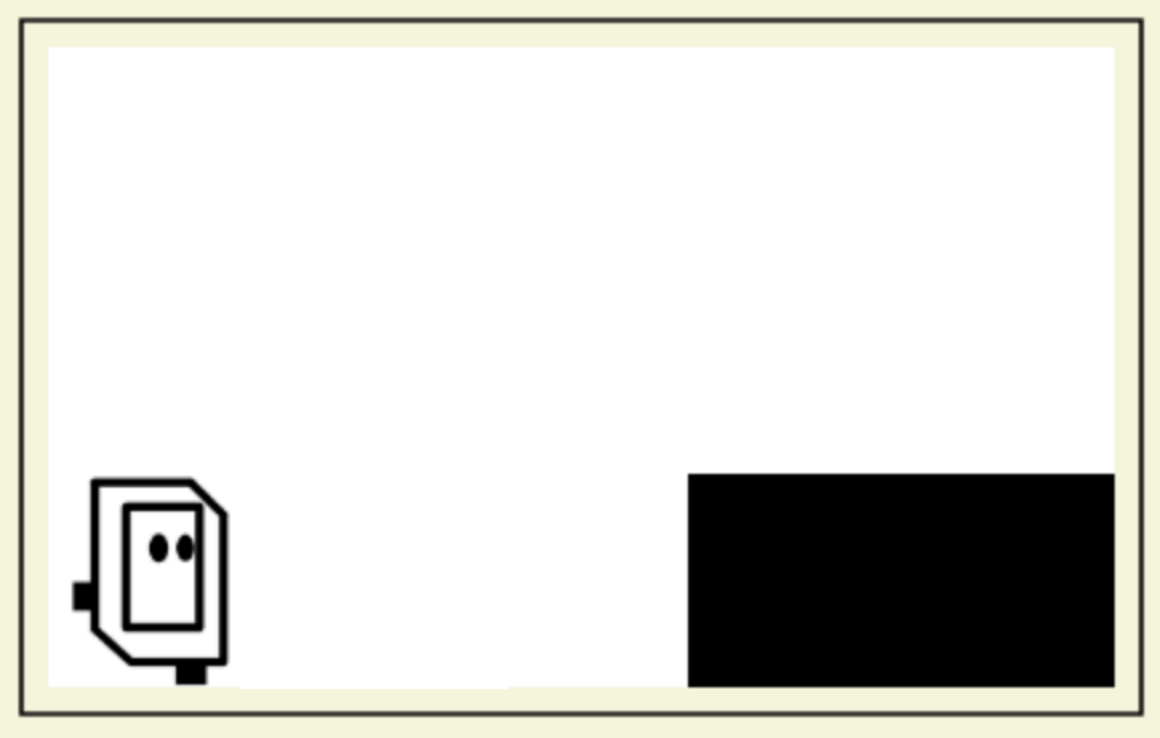
Meet Bit

- Bit cannot move past the end of the grid (the outer walls), or there is an error.



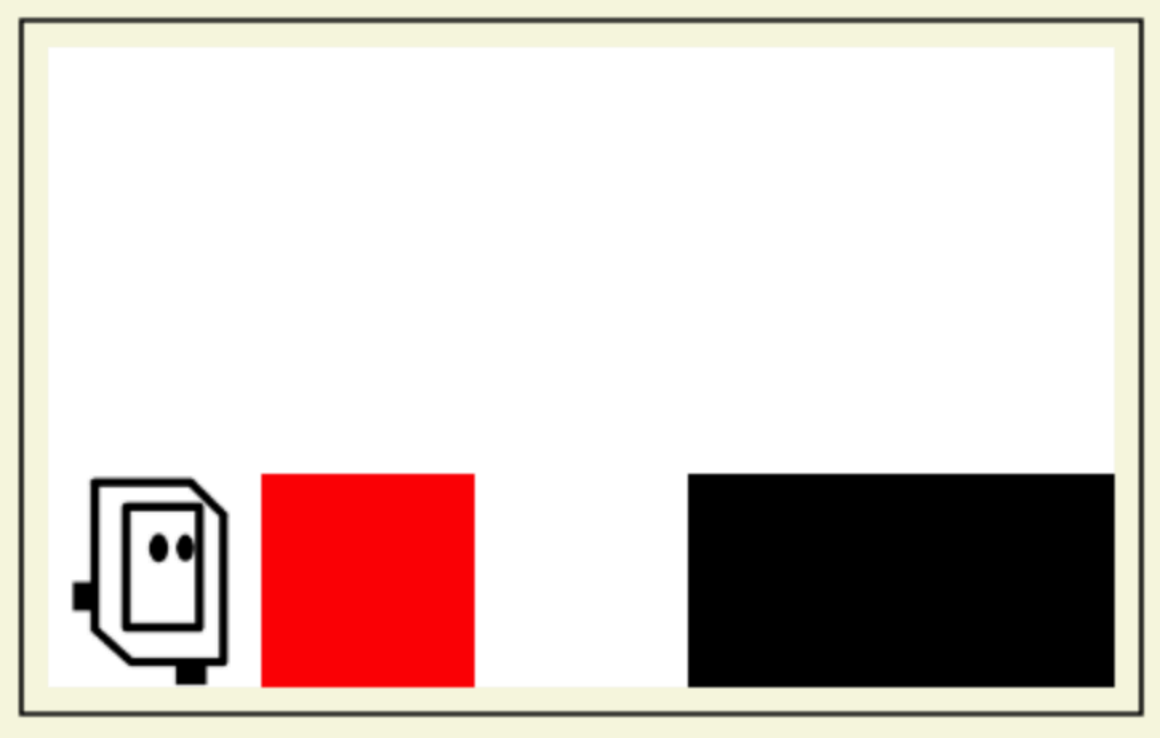
Meet Bit

- Bit cannot move past the end of the grid (the outer walls), or there is an error.
- There might also be inner walls, which are black. Bit cannot move through those walls, either.



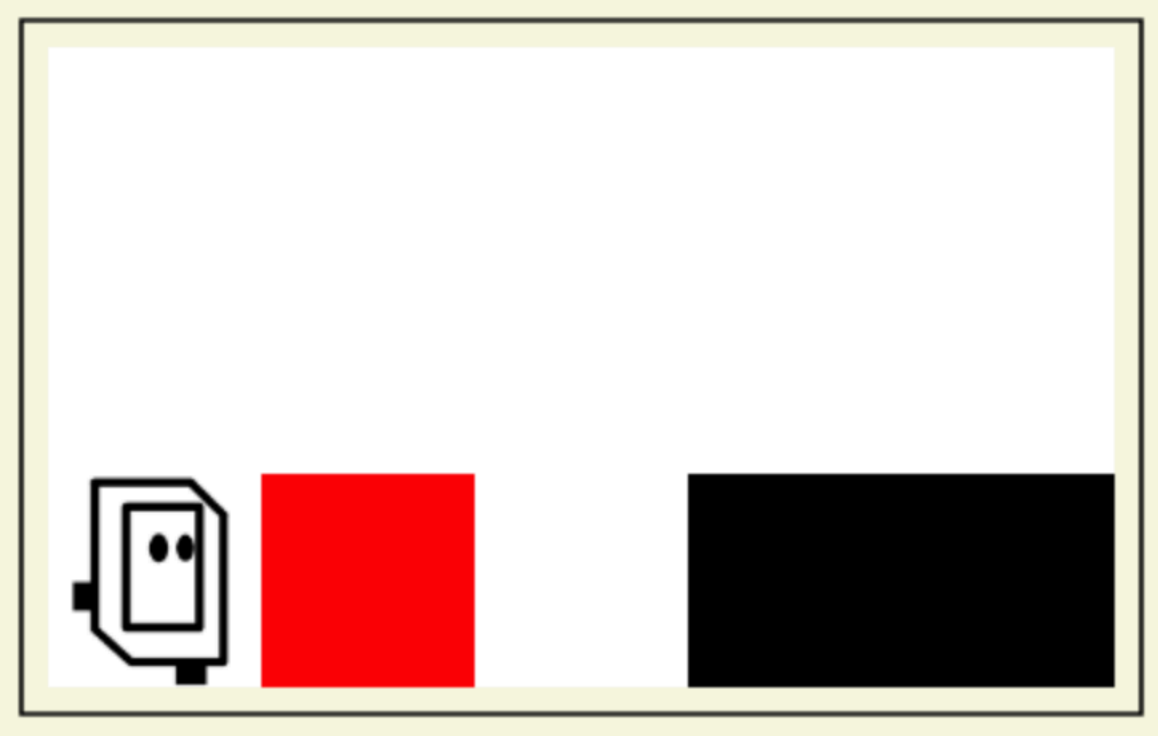
Meet Bit

- There might also be inner walls, which are black. Bit cannot move through those walls, either.
- Squares can be colored red, green, or blue, and Bit *can* move over those just fine (i.e., they are *not* walls).



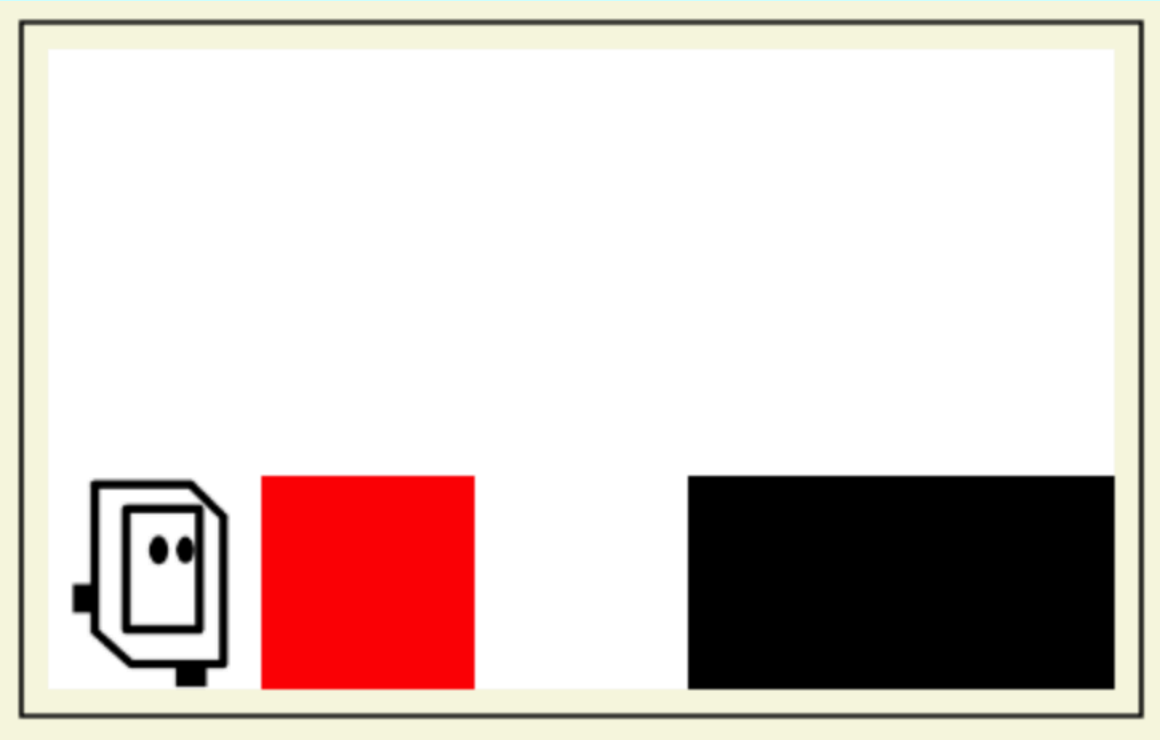
Meet Bit

- Bit starts out understanding a small number of action commands:
  - `bit.left()` turn left
  - `bit.right()` turn right
  - `bit.move()` move in the direction Bit is facing
  - `bit.paint(color)` paint the square `'red'`, `'green'`, or `'blue'`
  - `bit.erase()` clear the color under bit (back to white)



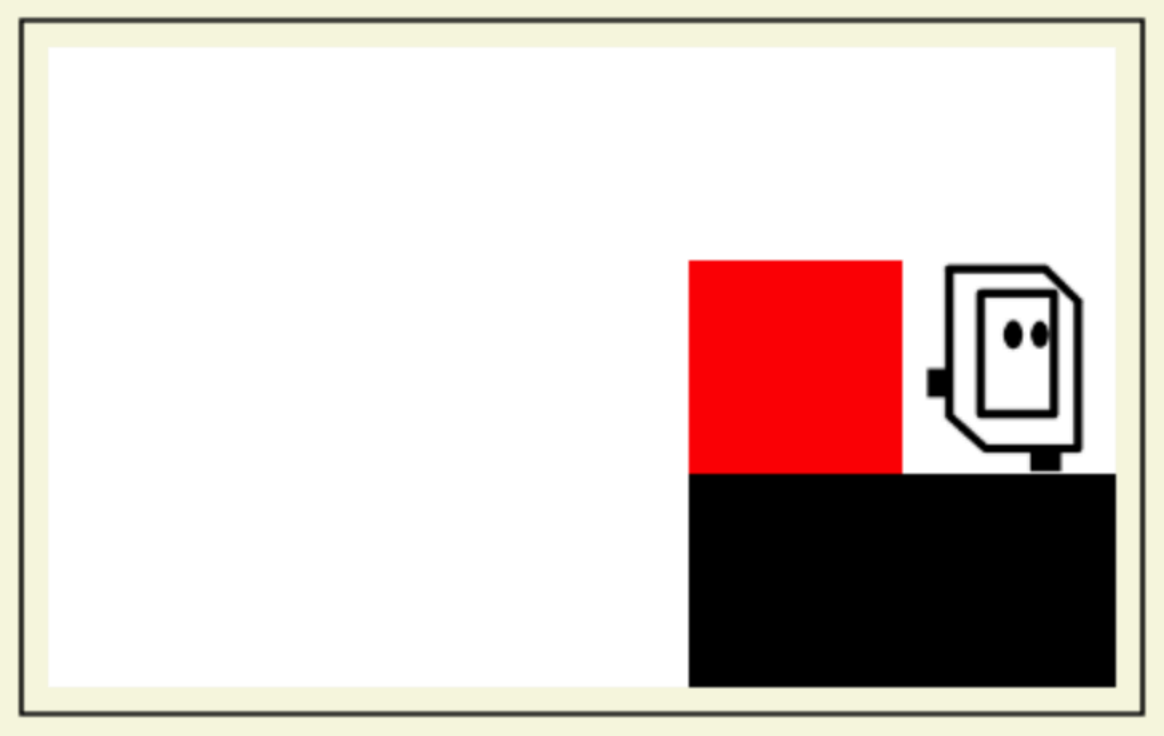
Your First Challenge

- How would you program Bit to erase the red square, and put a red square "on top" of the "ledge" (the black wall), with Bit ending up on the right side of the ledge, still facing right?



Your First Challenge

- How would you program Bit to erase the red square, and put a red square "on top" of the "ledge" (the black wall), with Bit ending up on the right side of the ledge, still facing right?



The `moveRedSquareToLedge` Function

```
// This program moves the red square up to a ledge.

function moveRedSquareToLedge(bit) {
    bit.move();
    bit.erase();
    bit.move();
    bit.left();
    bit.move();
    bit.right();
    bit.move();
    bit.paint('red');
    bit.move();
}
```

The `moveRedSquareToEdge` Function

Comment

```
// This program moves the red square up to a ledge.
function moveRedSquareToLedge(bit) {
    bit.move();
    bit.erase();
    bit.move();
    bit.left();
    bit.move();
    bit.right();
    bit.move();
    bit.paint('red');
    bit.move();
}
```

The `moveRedSquareToEdge` Function

Comment

```
// This program moves the red square up to a ledge.
function moveRedSquareToLedge(bit) {
    bit.move();
    bit.erase();
    bit.move();
    bit.left();
    bit.move();
    bit.right();
    bit.move();
    bit.paint('red');
    bit.move();
}
```

The program function

The `moveRedSquareToEdge` Function

Notice that the program on the prior slides is in Javascript — for CS106A, which is taught in Python, Bit understands Python:

```
# This program moves the red square up to a ledge.

def moveRedSquareToLedge(bit):
    bit.move()
    bit.erase()
    bit.move()
    bit.left()
    bit.move()
    bit.right()
    bit.move()
    bit.paint('red')
    bit.move()
```

Defining New Functions

- A Bit program consists of a collection of ***functions***, each of which is a sequence of statements that has been collected together and given a name. The pattern for defining a new function looks like this:

```
function name() {
    statements that implement the desired operation
}
```

- In patterns of this sort, the boldfaced words are fixed parts of the pattern; the italicized parts represent the parts you can change. Thus, every helper function will include the keyword **`function`** along with the parentheses and braces shown. You get to choose the name and the sequence of statements performs the desired operation.

Adding Functions to a Program

```
// This program moves the red squad up to a ledge.
// And then moves Bit back down again

function moveRedSquareToLedge(bit) {
    bit.move();
    bit.erase();
    bit.move();
    bit.left();
    bit.move();
    bit.right();
    bit.move();
    bit.paint('red');
    bit.move();
    turnAround(bit);
    moveBackDown(bit);
}

function turnAround(bit) {
    bit.right();
    bit.right();
}

function moveBackDown(bit) {
    bit.move();
    bit.move();
    bit.left();
    bit.move();
    bit.right();
    bit.move();
    bit.move();
    turnAround(bit);
}
```

Exercise: Defining Functions

- Define a function called **`turnAround`** that turns Bit around 180 degrees without moving.

```
function turnAround(bit) {
   bit.right();
   bit.right();
}
```

- Define a function **`backup`** that moves Karel backward one square, leaving Karel facing in the same direction.

```
function backup(bit) {
   turnAround(bit);
   bit.move();
   turnAround(bit);
}
```

Control Statements

- In addition to allowing you to define new functions, Bit also allows standard Javascript (or Python) control statements:
- The control statements available in Karel are:
  - The **`while`** statement, which repeats a set of statements as long as some condition holds.
  - The **`if`** statement, which applies a conditional test to determine whether a set of statements should be executed at all.
  - The **`if-else`** statement, which uses a conditional test to choose between two possible actions.

Conditions in Bit

- Bit can test the following conditions:

`bit.front_clear()`
`bit.left_clear()`
`bit.right_clear()`
`bit.get_color()`

- The first three conditions can be used to tell whether there is a wall in front of, or to the left/right of Bit. These are useful to continue walking in a direction until a wall appears, or a wall begins or ends.
- The `bit.get_color()` function returns either `'red'`, `'green'`, `'blue'`, or `null`, depending on what color is at Bit's position.

The `while` Statement

- The general form of the **`while`** statement looks like this:

```
while (condition) {
    statements to be repeated
}
```

- The simplest example of the **`while`** statement is the function **`moveToWall`**, which comes in handy in lots of programs:

```
function moveToWall() {
   while (bit.front_clear()) {
      bit.move();
   }
}
```

The `if` and `if-else` Statements

- The **`if`** statement in Bit comes in two forms:
  - A simple **`if`** statement for situations in which you may or may not want to perform an action:

```
if (condition) {
    statements to be executed if the condition is true
}
```

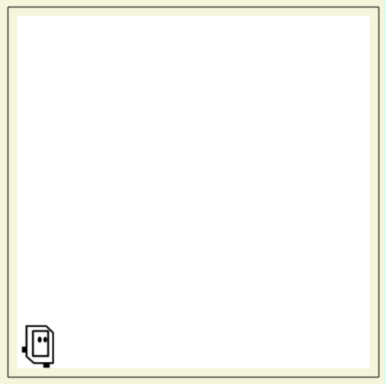
  - An **`if-else`** statement for situations in which you must choose between two different actions:

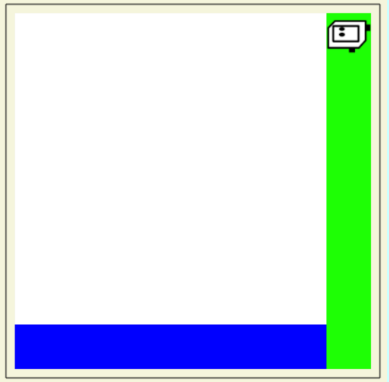
```
if (condition) {
    statements to be executed if the condition is true
} else {
    statements to be executed if the condition is false
}
```

Exercise: Creating a Green Line

- Write a function **`colorLine(color)`** that colors each square up to a wall in the direction Bit is traveling.
- Your function should operate correctly no matter how far Bit is from the wall or what direction Bit is facing.
- Consider, for example, the following function called **`test`**:

```
function test(bit) {
   colorLine(bit, 'blue');
   bit.left();
   colorLine(bit, 'green');
}
```



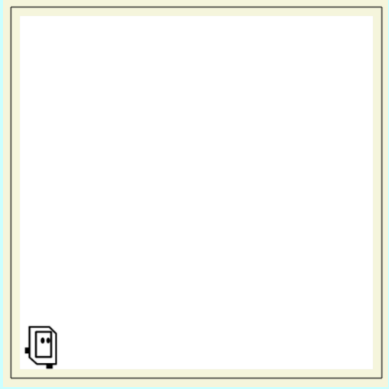


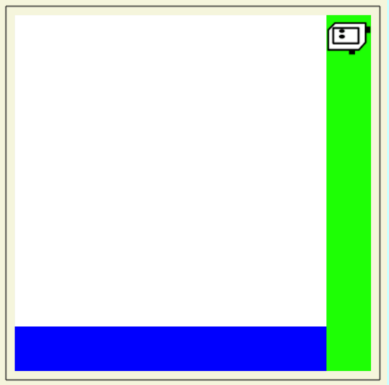
Exercise: Creating a Green Line

```
function test(bit) {
    colorLine(bit, 'blue');
    bit.left();
    colorLine(bit, 'green');
}

function colorLine(bit, color)
{
    bit.paint(color);
    while (bit.front_clear()) {
        bit.move();
        bit.paint(color);
    }

}
```





Pascal

```
program HelloWorld;

var                                    https://onlinegdb.com/G_g_tcAHt
   i: integer;
   s: string;

begin
    i := 10;
    repeat
        str(i, s);
        Writeln('Hello world! ' + s);
        i := i - 1;
    until i = 0;
end.
```

Pascal was invented by (future) Turing Award winner and former Stanford professor, Niklaus Wirth in 1970 (a few years after he left Stanford).

Wirth designed it to be a small language that encouraged good style, and because of this it was used in many universities (including Stanford) in the 1970s and 1980s (and eventually generally replaced by C).

Pascal

https://onlinegdb.com/OCc9XJW57

```
program ReverseString;

function revstr(my_s:string):string;
    var
        out_s: string;
        ls, i: integer;
    begin
        out_s := '';
        ls:=length(my_s);
        for i:=1 to ls do
            out_s:=out_s+my_s[ls-i+1];
        revstr:=out_s;
    end;

var
        original, reversed: string;

begin
    original := 'Hello World';
    reversed := revstr(original);
    Writeln('Original: ' + original);
    Writeln('Reversed: ' + reversed);
end.
```

Pascal had some interesting features: notably:

- variable assignment used ":=" instead of "=", allowing for a more beginner-friendly syntax
- string lengths were part of the string type (and strings were not 0-terminated, but rather had their length embedded in the string). This was a problem, and necessitated changing the language, eventually (e.g., this made it almost impossible to write a sorting library).

Pascal

https://onlinegdb.com/xVT592imt

```
function factorial(n: integer): integer;
 begin
   if n = 0
     then
       factorial := 1
     else
       factorial := n*factorial(n-1)
 end;

 var
   number: integer;

 begin
    number := 15;
    Writeln(factorial(number));
 end.
```

Pascal gained a huge following in the mid-1980s when Turbo Pascal by Borland was released. It was inexpensive, and came with a built-in Integrated Development Environment (IDE) that allowed programmers an easy way to write programs that compiled to machine code, and that were extremely fast (see the Wikipedia article for an interesting anecdote about Bill Gates).

Turbo Pascal also shipped on a single 360KB floppy disk, meaning it could be used on virtually any 1980s vintage PC.

BASIC

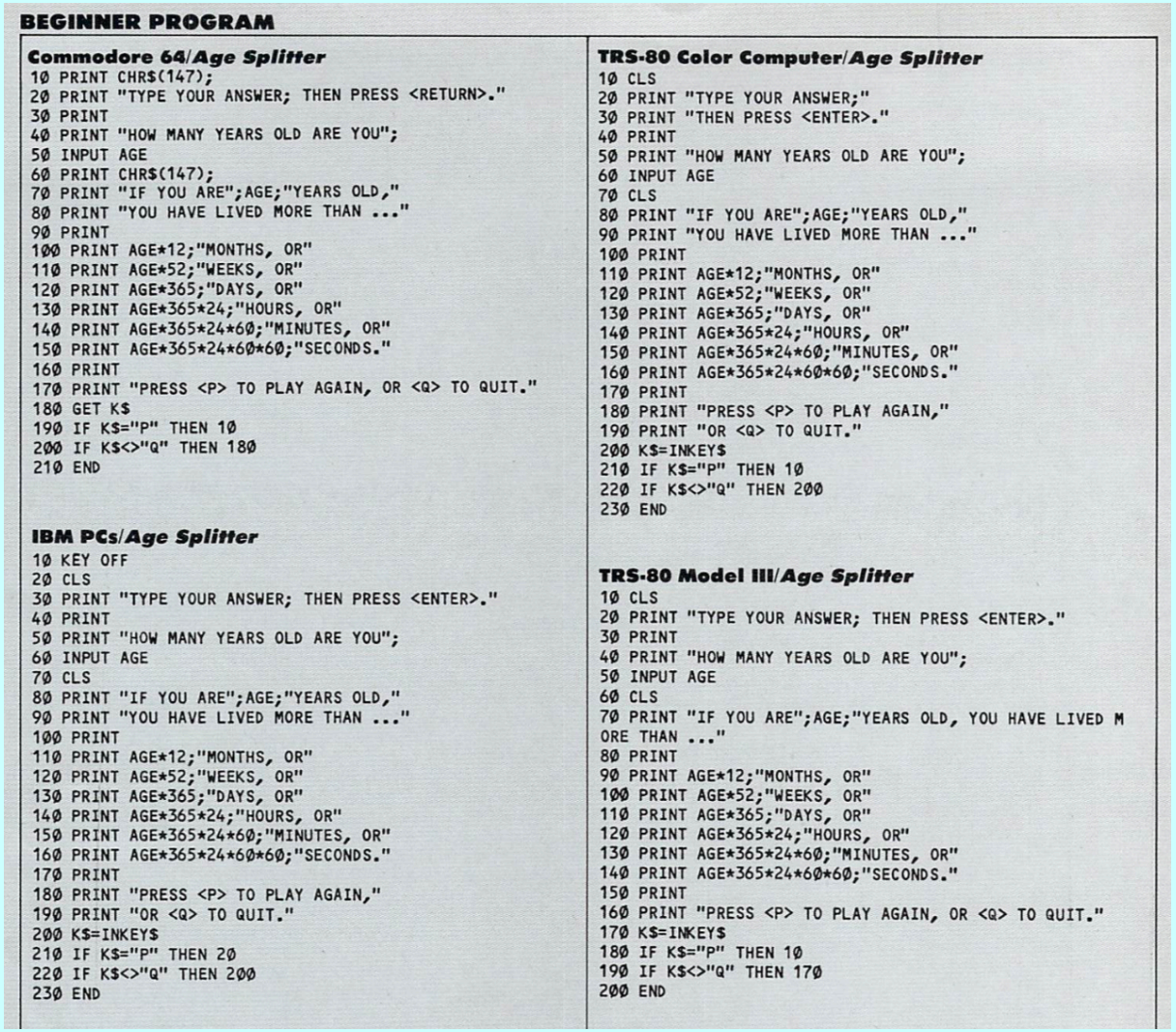
The BASIC language, created in 1964 by John G. Kemeny and Thomas E. Kurtz at Dartmouth, was *the* programming language that was included with home computers during the 1970s and 1980s.

The language was usually included in ROM, so that when the computer booted up, users could immediately start programming.

BASIC

BASIC was designed so that students in non-scientific fields could learn to program.

Students learned BASIC in school, or by reading books that had listing of programs (often games) that they could type in relatively quickly.



BEGINNER PROGRAM

Commodore 64/*Age Splitter*

```
10 PRINT CHR$(147);
20 PRINT "TYPE YOUR ANSWER; THEN PRESS <RETURN>."
30 PRINT
40 PRINT "HOW MANY YEARS OLD ARE YOU";
50 INPUT AGE
60 PRINT CHR$(147);
70 PRINT "IF YOU ARE";AGE;"YEARS OLD,"
80 PRINT "YOU HAVE LIVED MORE THAN ..."
90 PRINT
100 PRINT AGE*12;"MONTHS, OR"
110 PRINT AGE*52;"WEEKS, OR"
120 PRINT AGE*365;"DAYS, OR"
130 PRINT AGE*365*24;"HOURS, OR"
140 PRINT AGE*365*24*60;"MINUTES, OR"
150 PRINT AGE*365*24*60*60;"SECONDS."
160 PRINT
170 PRINT "PRESS <P> TO PLAY AGAIN, OR <Q> TO QUIT."
180 GET K$
190 IF K$="P" THEN 10
200 IF K$<>"Q" THEN 180
210 END
```

IBM PCs/*Age Splitter*

```
10 KEY OFF
20 CLS
30 PRINT "TYPE YOUR ANSWER; THEN PRESS <ENTER>."
40 PRINT
50 PRINT "HOW MANY YEARS OLD ARE YOU";
60 INPUT AGE
70 CLS
80 PRINT "IF YOU ARE";AGE;"YEARS OLD,"
90 PRINT "YOU HAVE LIVED MORE THAN ..."
100 PRINT
110 PRINT AGE*12;"MONTHS, OR"
120 PRINT AGE*52;"WEEKS, OR"
130 PRINT AGE*365;"DAYS, OR"
140 PRINT AGE*365*24;"HOURS, OR"
150 PRINT AGE*365*24*60;"MINUTES, OR"
160 PRINT AGE*365*24*60*60;"SECONDS."
170 PRINT
180 PRINT "PRESS <P> TO PLAY AGAIN,"
190 PRINT "OR <Q> TO QUIT."
200 K$=INKEY$
210 IF K$="P" THEN 20
220 IF K$<>"Q" THEN 200
230 END
```

TRS-80 Color Computer/*Age Splitter*

```
10 CLS
20 PRINT "TYPE YOUR ANSWER;"
30 PRINT "THEN PRESS <ENTER>."
40 PRINT
50 PRINT "HOW MANY YEARS OLD ARE YOU";
60 INPUT AGE
70 CLS
80 PRINT "IF YOU ARE";AGE;"YEARS OLD,"
90 PRINT "YOU HAVE LIVED MORE THAN ..."
100 PRINT
110 PRINT AGE*12;"MONTHS, OR"
120 PRINT AGE*52;"WEEKS, OR"
130 PRINT AGE*365;"DAYS, OR"
140 PRINT AGE*365*24;"HOURS, OR"
150 PRINT AGE*365*24*60;"MINUTES, OR"
160 PRINT AGE*365*24*60*60;"SECONDS."
170 PRINT
180 PRINT "PRESS <P> TO PLAY AGAIN,"
190 PRINT "OR <Q> TO QUIT."
200 K$=INKEY$
210 IF K$="P" THEN 10
220 IF K$<>"Q" THEN 200
230 END
```

TRS-80 Model III/*Age Splitter*

```
10 CLS
20 PRINT "TYPE YOUR ANSWER; THEN PRESS <ENTER>."
30 PRINT
40 PRINT "HOW MANY YEARS OLD ARE YOU";
50 INPUT AGE
60 CLS
70 PRINT "IF YOU ARE";AGE;"YEARS OLD, YOU HAVE LIVED M
ORE THAN ..."
80 PRINT
90 PRINT AGE*12;"MONTHS, OR"
100 PRINT AGE*52;"WEEKS, OR"
110 PRINT AGE*365;"DAYS, OR"
120 PRINT AGE*365*24;"HOURS, OR"
130 PRINT AGE*365*24*60;"MINUTES, OR"
140 PRINT AGE*365*24*60*60;"SECONDS."
150 PRINT
160 PRINT "PRESS <P> TO PLAY AGAIN, OR <Q> TO QUIT."
170 K$=INKEY$
180 IF K$="P" THEN 10
190 IF K$<>"Q" THEN 170
200 END
```

Family Computing Magazine, July 1985

BASIC

While BASIC was relatively easy to learn, and while it introduced millions of kids to programming, it is not a particularly good language (at least the 1980s version — today, Visual Basic is decent).

Famously, Edsgar Dijkstra said, in 1975, "It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration."

Students who learned BASIC on their own do, indeed, have some trouble graduating to a *structured* language such as C, Java, Javascript, etc., but it is probably not as dire as Dijkstra led on.

The End