/coding

Code and Space
ARC 593 | DMS 606

Fall 2016
Instructor: Mark Shepard
TA: Leonardo Aranda

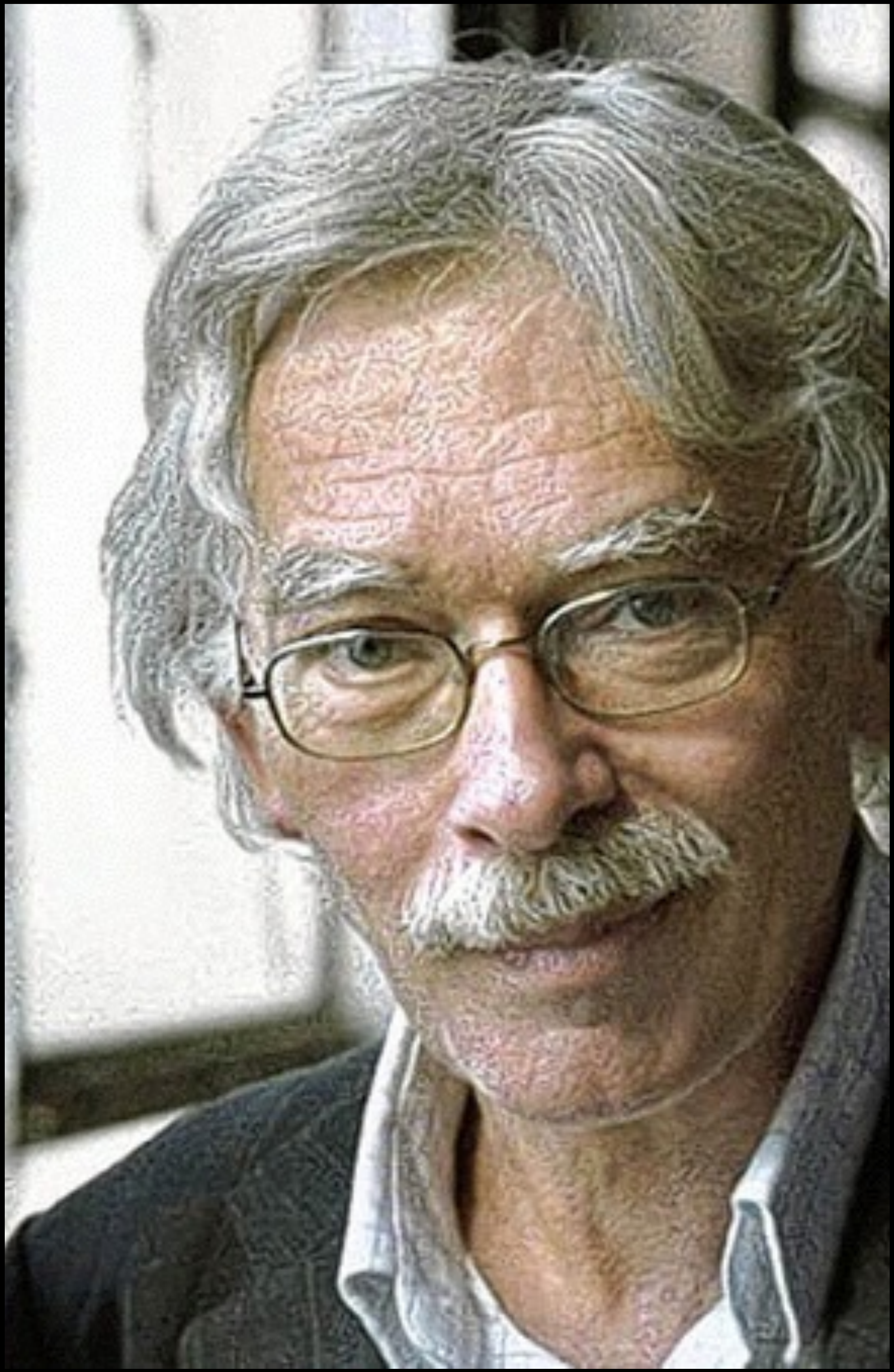
what is code?

The modern city exists as a haze of software instructions. (Amin and Thrift)

Values, opinions and rhetoric are frozen into code. (Bowker and Leigh-Star)

Frederich Kittler

Code, or How to Write Something Differently



codes…

are the language of our time

- codes materialize the process of encryption
- Wolfgang Coy: “a mapping of a finite set of symbols of an alphabet onto a suitable signal sequence”

codes…

are not what what we may think:

- codes are not specific to computer technology or genetic engineering
- as sequences of signals over time, they are part of every communications technology

codes…

became conceivable/feasible given:

- alphabets that mapped speech sounds onto letters (more or less one-to-one)
- developed communications technology (secret messages, encryption)

codes…

command, code and communications

- command, code and communications technologies are brought together by the Empire

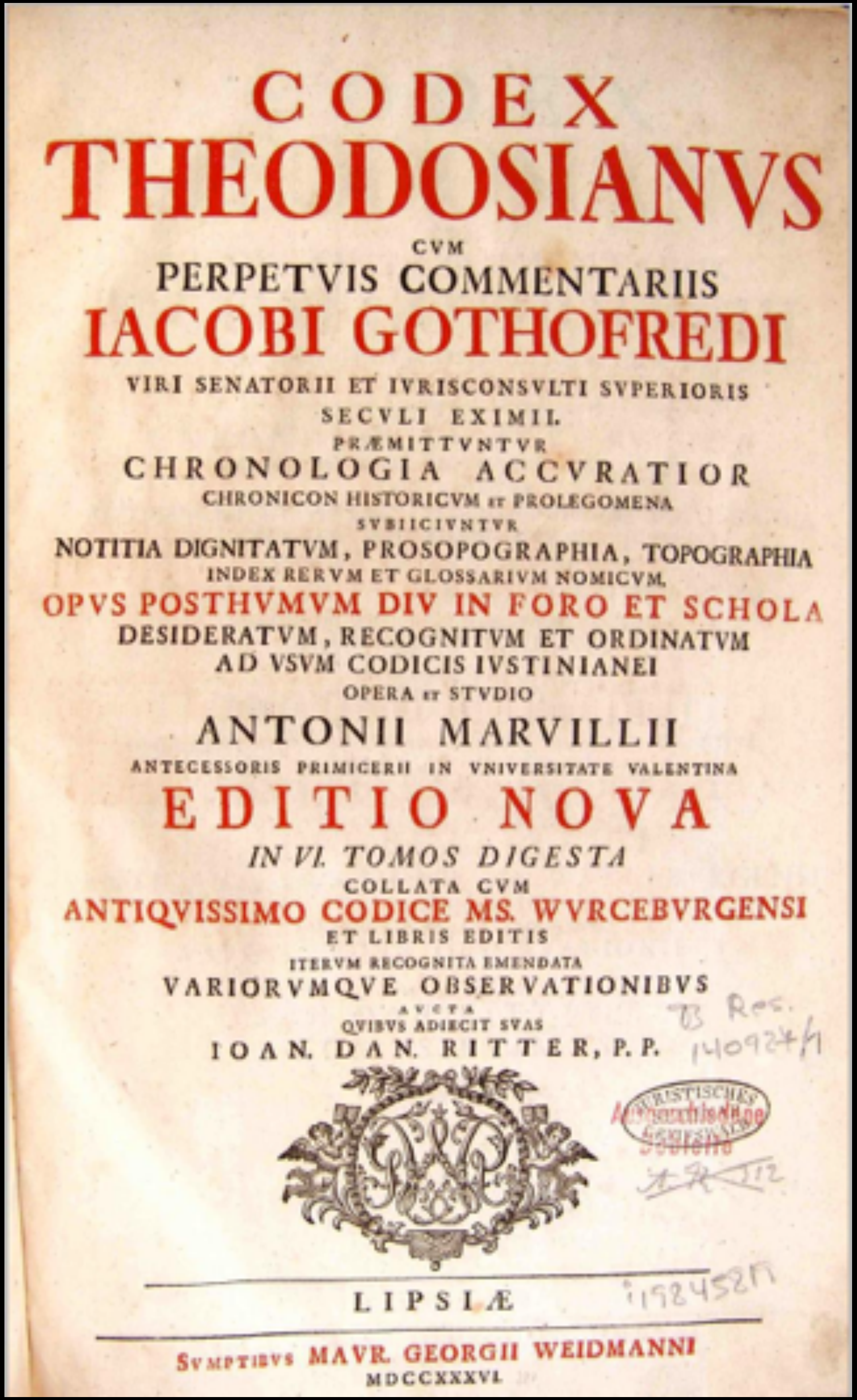
- code as the bound book of law

- codification as the judicial-bureaucratic act collecting imperial commands into a single collection of laws

- message transmission > data storage

codes…

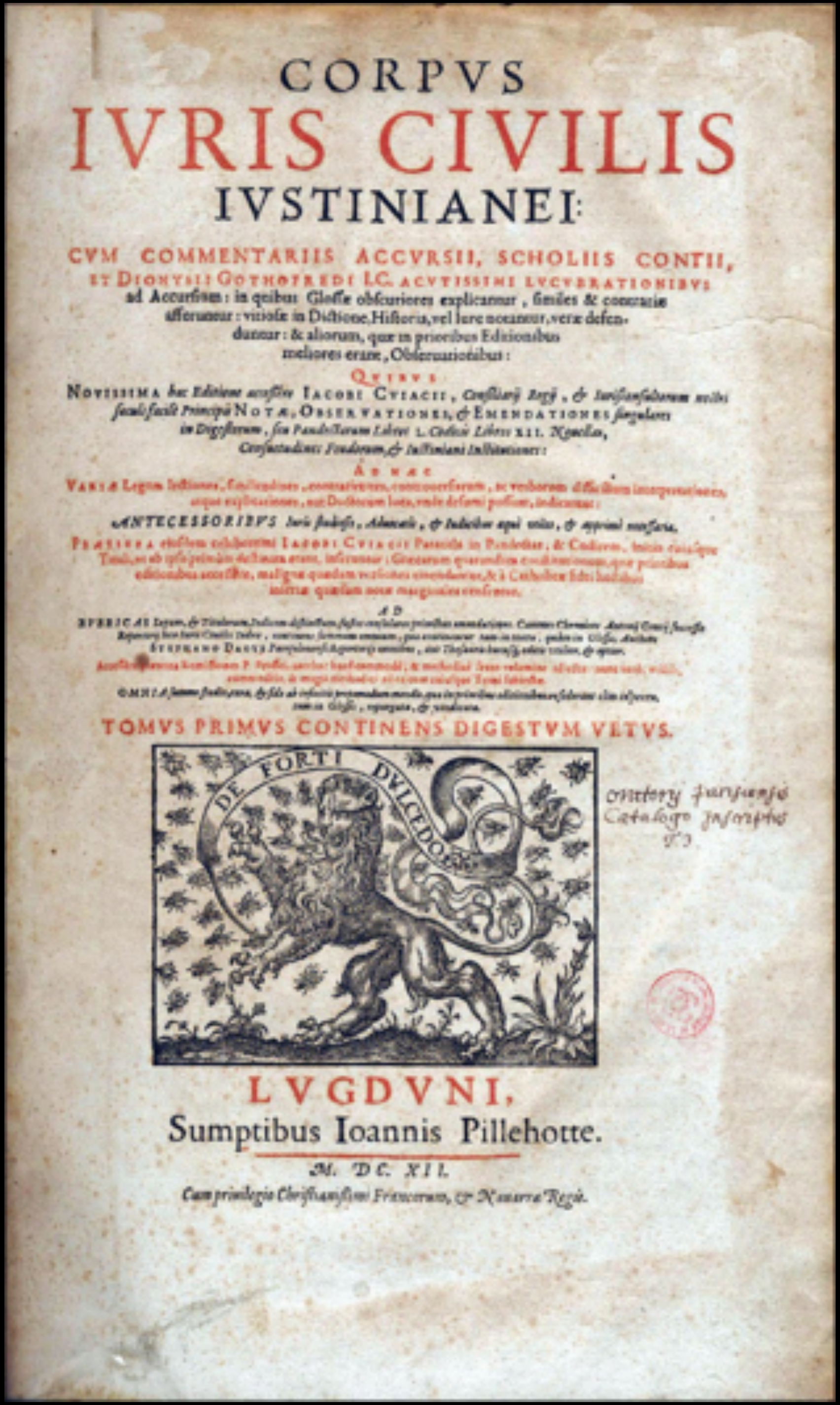
the codex



CODEX
THEODOSIANVS
CVM
PERPETVIS COMMENTARIIS
IACOBI GOTHOFREDI
VIRI SENATORII ET IVRISCONSVLTI SVPERIORIS
SECVLI EXIMII.
PRÆMITTVNTVR
CHRONOLOGIA ACCVRATIOR
CHRONICON HISTORICVM ET PROLEGOMENA
SVBIICIVNTVR
NOTITIA DIGNITATVM, PROSOPOGRAPHIA, TOPOGRAPHIA
INDEX RERVM ET GLOSSARIVM NOMICVM.
OPVS POSTHVMVM DIV IN FORO ET SCHOLA
DESIDERATVM, RECOGNITVM ET ORDINATVM
AD VSVM CODICIS IVSTINIANEI
OPERA ET STVDIO
ANTONII MARVILLII
ANTECESSORIS PRIMICERII IN VNIVERSITATE VALENTINA
EDITIO NOVA
IN VI. TOMOS DIGESTA
COLLATA CVM
ANTIQVISSIMO CODICE MS. WVRCEBVRGENSI
ET LIBRIS EDITIS
ITERVM RECOGNITA EMENDATA
VARIORVMQVE OBSERVATIONIBVS
AVCTA
QVIBVS ADIECIT SVAS
IOAN. DAN. RITTER, P. P.

LIPSIÆ

SVMPTIBVS MAVR. GEORGII WEIDMANNI
MDCCXXXVI.



CORPVS
IVRIS CIVILIS
IVSTINIANEI:
CVM COMMENTARIIS ACCVRSII, SCHOLIIS CONTII,
ET DIONYSII GOTHOFREDI I.C. ACVTISSIMI LVCVBRATIONIBVS
ad Accursium: in quibus Glossæ obscuriores explicantur, similes & contrariæ afferuntur: vitiosæ in Dictione, Historia, vel Iure notantur, veræ defenduntur: & aliorum, quæ in prioribus Editionibus meliores erant, Obseruationibus:
QVIBVS
NOVISSIMA *hac Editione accessere* IACOBI CVIACII, *Consiliarij Regij, & Iurisconsultorum nostri sæculi facilè Principis* NOTÆ, OBSERVATIONES, & EMENDATIONES *singulares in Digestorum, seu Pandectarum Libros L. Codicis Libros* XII. *Nouellas, Consuetudines Feudorum, & Iustiniani Institutiones:*
AD HÆC
VARIÆ Legum lectiones, [illegible]
ANTECESSORIBVS Iuris studiosis, Aduocatis, & Iudicibus [illegible] *& apprimè necessaria.*
PRÆTEREA eiusdem celeberrimi IACOBI CVIACII [illegible]
AD
RVBRICAS [illegible]
STEPHANO DAOYZ [illegible]
[illegible]
OMNIA [illegible]
TOMVS PRIMVS CONTINENS DIGESTVM VETVS.

Oratorij Parisiensis Catalogo inscriptus

LVGDVNI,
Sumptibus Ioannis Pillehotte.
M. DC. XII.
Cum priuilegio Christianissimi Francorum, & Nauarræ Regis.

codes…

morse code

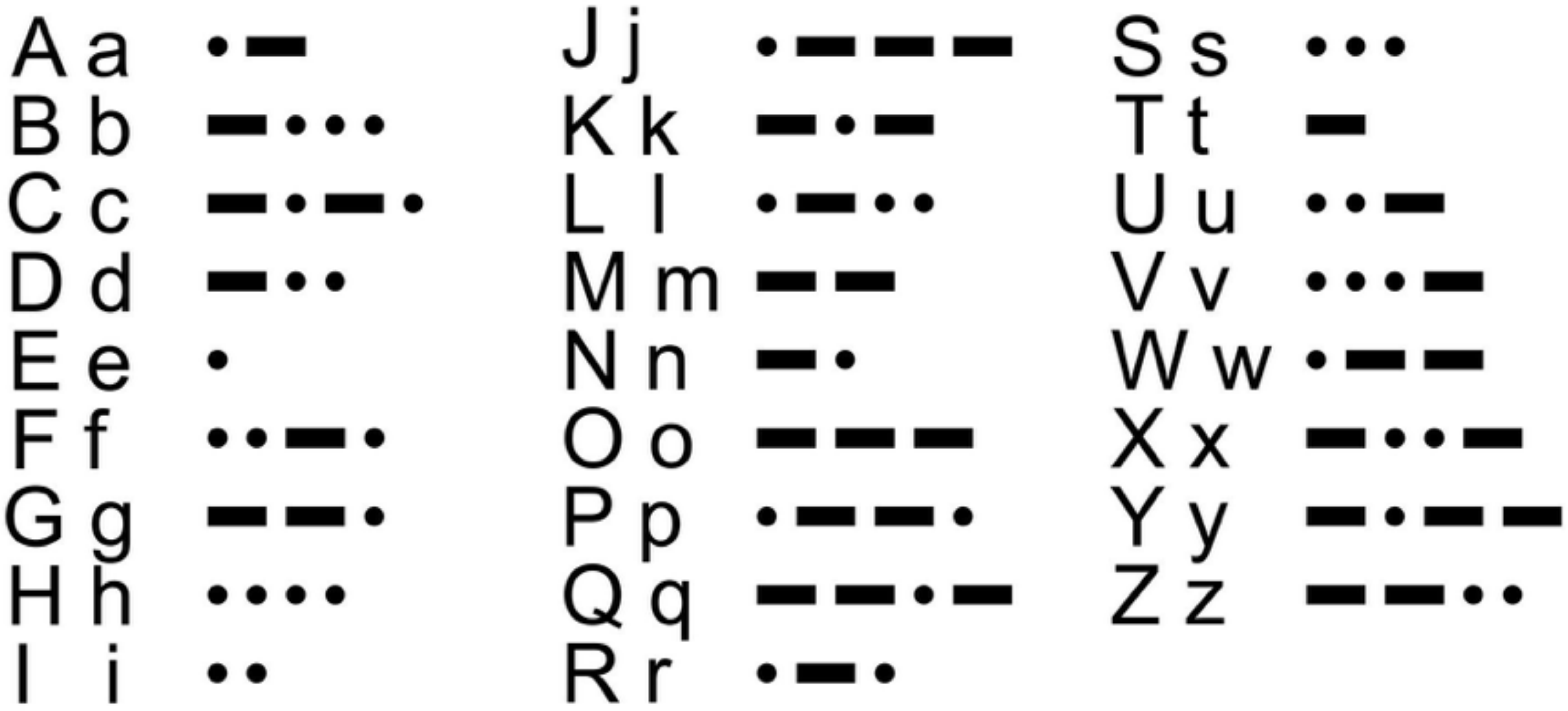
- a system of writing optimized according to technical criteria (no regard to semantics)

- from deciphering / enciphering to encoding / decoding

- code is elegant if its output is longer than itself

codes…

morse code



A a	•▬	J j	•▬▬▬	S s	•••
B b	▬•••	K k	▬•▬	T t	▬
C c	▬•▬•	L l	•▬••	U u	••▬
D d	▬••	M m	▬▬	V v	•••▬
E e	•	N n	▬•	W w	•▬▬
F f	••▬•	O o	▬▬▬	X x	▬••▬
G g	▬▬•	P p	•▬▬•	Y y	▬•▬▬
H h	••••	Q q	▬▬•▬	Z z	▬▬••
I i	••	R r	•▬•		

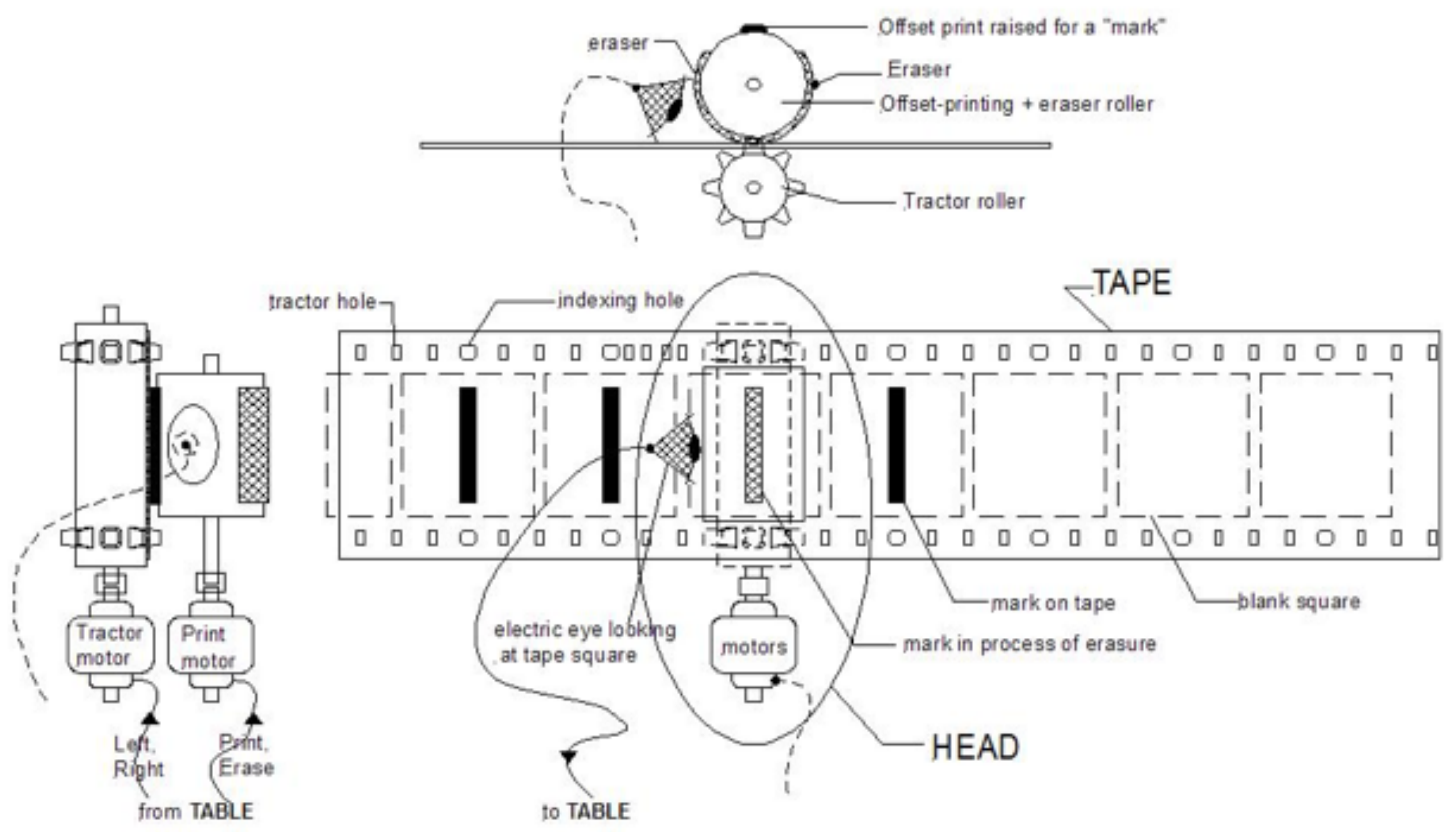
codes…

The Turing Machine (1936)

- convergence of mathematics and encryption
- an abstract machine that manipulates symbols on a strip of tape according to a table of rules
- a mathematical model of computation that defines such a device

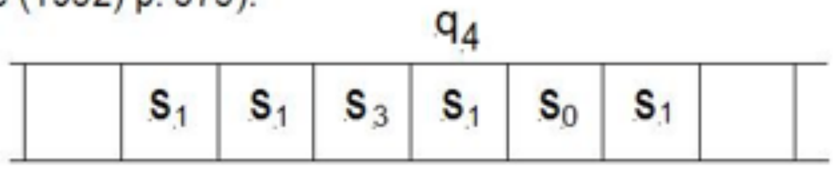
codes…

The Turing Machine (1936)



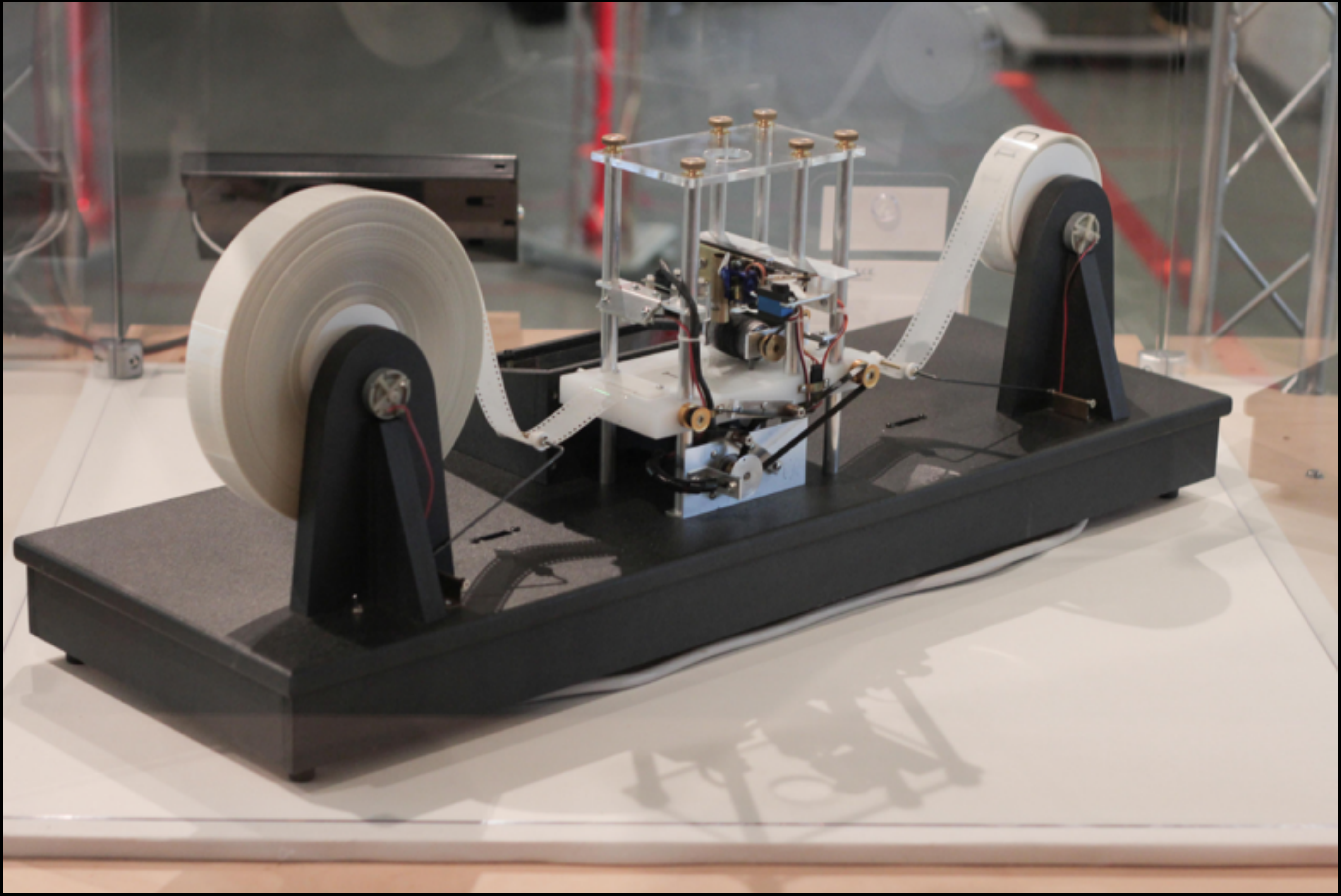
Above: a fanciful mechanical Turing machine's TAPE and HEAD (TABLE not shown).

Below: Usually Turing machines are drawn in a much simpler way -- as just a row of squares with symbols in them, the "head" drawn over the "square being scanned" (drawing after Kleene (1952) p. 375).



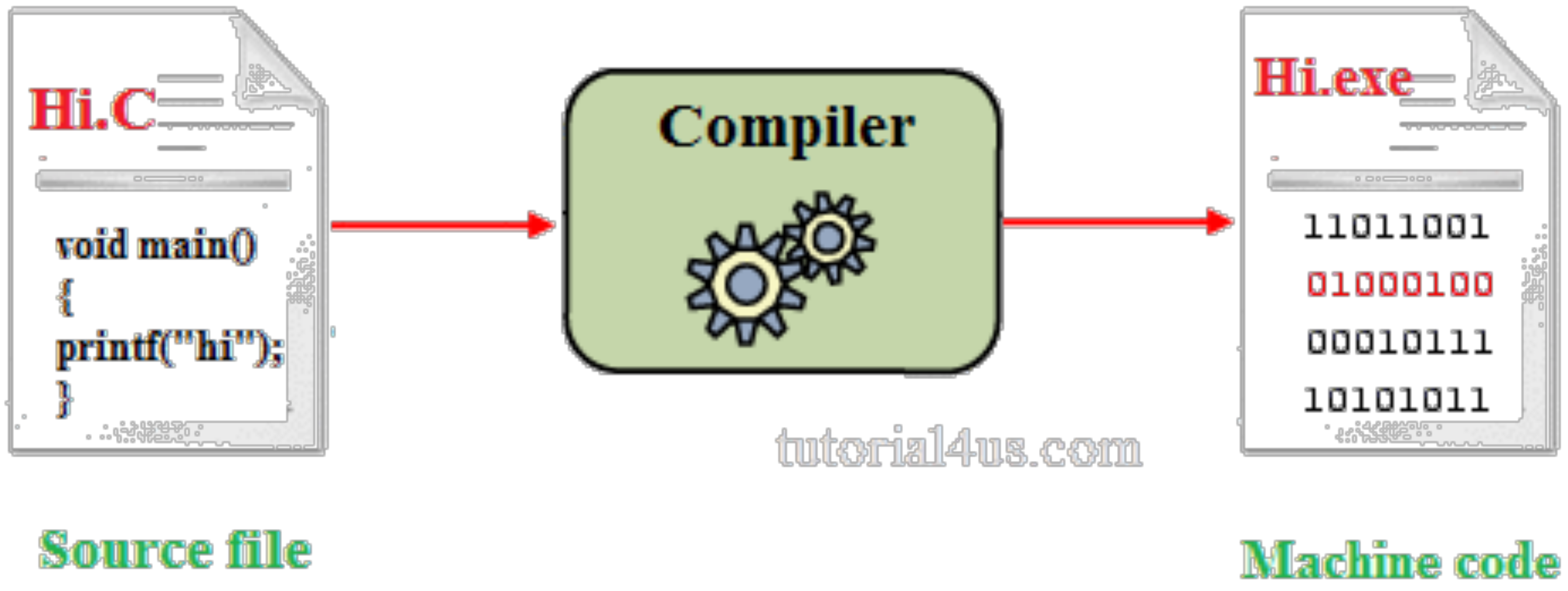
codes…

The Turing Machine (1936)



codes…

from high to low level language



```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */


int ledPin = 13;                // LED connected to digital pin 13

void setup()                    // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}


void loop()                     // run over and over again
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Comments

```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */

int ledPin = 13;                // LED connected to digital pin 13

void setup()                    // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}


void loop()                     // run over and over again
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */
```

Declare global variables

```
int ledPin = 13;                // LED connected to digital pin 13

void setup()                    // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}


void loop()                     // run over and over again
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */


int ledPin = 13;                // LED connected to digital pin 13
```

Initial function call

```
void setup()                    // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}


void loop()                     // run over and over again
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */


int ledPin = 13;                // LED connected to digital pin 13

void setup()                    // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}
```

Main program function, called infinitely

```
void loop()                     // run over and over again
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Comments

Comments are notes to you (or others trying to understand your code) that are ignored by the program interpreter.

```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */
```

```
// this is a single line comment

/* this is how you start a multiline comment
and this is how you end it */
```

Anything between the first `/*` and the next `*/` will be ignored

Variables

Variables are places to store data you use more than once.

```
int ledPin = 13;                  // LED connected to digital pin 13
```

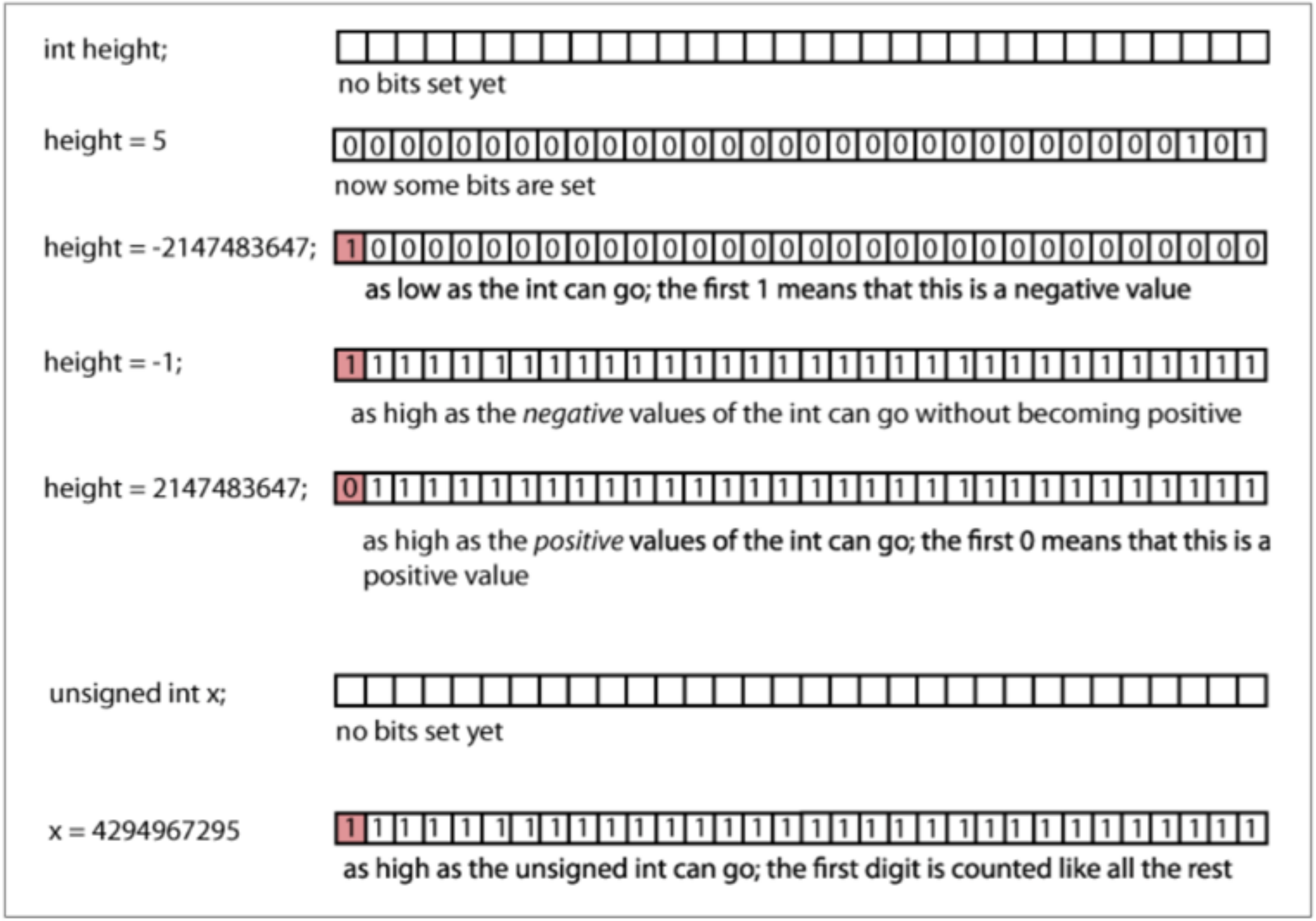
box type	**box name**	**=**	**box contents**
`int`	`ledPin`	`=`	`13`

The first part of the line before the comment is called a **statement**. All statements end with a semicolon **`;`**

Think of variables as boxes where you store different kinds of data. In this case, **`int`** means "integer" or simply a whole number.

This **statement** creates a box called **`ledPin`** and stores the integer **13** in this box.

Variables



int height;
no bits set yet
height = 5
00000000000000000000000000000101
now some bits are set
height = -2147483647;
10000000000000000000000000000000
as low as the int can go; the first 1 means that this is a negative value
height = -1;
11111111111111111111111111111111
as high as the *negative* values of the int can go without becoming positive
height = 2147483647;
01111111111111111111111111111111
as high as the *positive* values of the int can go; the first 0 means that this is a positive value
unsigned int x;
no bits set yet
x = 4294967295
11111111111111111111111111111111
as high as the unsigned int can go; the first digit is counted like all the rest

Figure 2-1. Setting the bits of signed and unsigned variables

Data types

Different kinds of data need different kinds of boxes.
Here are some different kinds of boxes you can use:

- `void`
- `boolean`
- `char`
- `unsigned char`
- `byte`
- `int`
- `unsigned int`
- `word`
- `long`
- `unsigned long`
- `float`
- `double`
- `string - char array`
- `String - object`
- `array`

(Click on each data type for more detail.)

Array

An array is a collection of variables that are accessed with an index number.

```
  int myInts[6];
  int myPins[] = {2, 4, 8, 3, 6};
  int mySensVals[6] = {2, 4, -8, 3, 2};
  char message[6] = "hello";
```

You can **declare** an array without initializing it as in myInts.

In myPins we declare an array without explicitly choosing a size. The compiler counts the elements and creates an array of the appropriate size.

Finally you can both initialize and size your array, as in mySensVals. Note that when declaring an array of type char, one more element than your initialization is required, to hold the required null character.

Array

Arrays are zero indexed, that is, referring to the array initialization above, the first element of the array is at index 0, hence:

```
int mySensVals[6] = {2, 4, -8, 3, 2};
mySensVals[0] == 2, mySensVals[1] == 4, […]
```

It also means that in an array with ten elements, index nine is the last element. So:

```
 int myArray[10]={9,3,2,4,3,2,7,8,9,11};
   // myArray[9]    contains 11
   // myArray[10]   is invalid and contains random information
                    (other memory address)
```

To assign a value to an array:

```
mySensVals[0] = 10;
```

To retrieve a value from an array:

```
x = mySensVals[4];
```

Control Structures

Control structures enable you to execute different blocks of code depending on different conditions - such as if a switch is on or off, or if a sensor exceeds a certain threshold, for example.

Here are some control structures you can use in Arduino:

- `if`
- `if...else`
- `for`
- `switch case`
- `while`
- `do... while`
- `break`
- `continue`
- `return`
- `goto`

(Click on each data type for more detail.)

if statement

If a (condition) is true, then execute the following statement(s). If not, do nothing.

```
if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);       // turn LED on:
  }
```

control	(condition)	{statements}
`if`	`(buttonState == HIGH)`	`{ digitalWrite(ledPin, HIGH); }`

Here are comparison operators you can use in your conditional statements (click for more info):

- `==` (equal to)
- `!=` (not equal to)
- `<` (less than)
- `>` (greater than)
- `<=` (less than or equal to)
- `>=` (greater than or equal to)

if ... else

```
int buttonPin = 2;      // the number of the pushbutton pin
int ledPin =  13;       // the number of the LED pin
int buttonState = 0;    // variable for reading the pushbutton status

void setup() {
  pinMode(ledPin, OUTPUT);    // initialize the LED pin as an output
  pinMode(buttonPin, INPUT); // initialize the pushbutton pin as an input
}

void loop(){
  buttonState = digitalRead(buttonPin); // read the state of the pushbutton
  // check if the pushbutton is pressed - if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);       // turn LED on:
  }
  else {
    digitalWrite(ledPin, LOW);        // turn LED off:
  }
}

// adapted from http://www.arduino.cc/en/Tutorial/Button
```

for

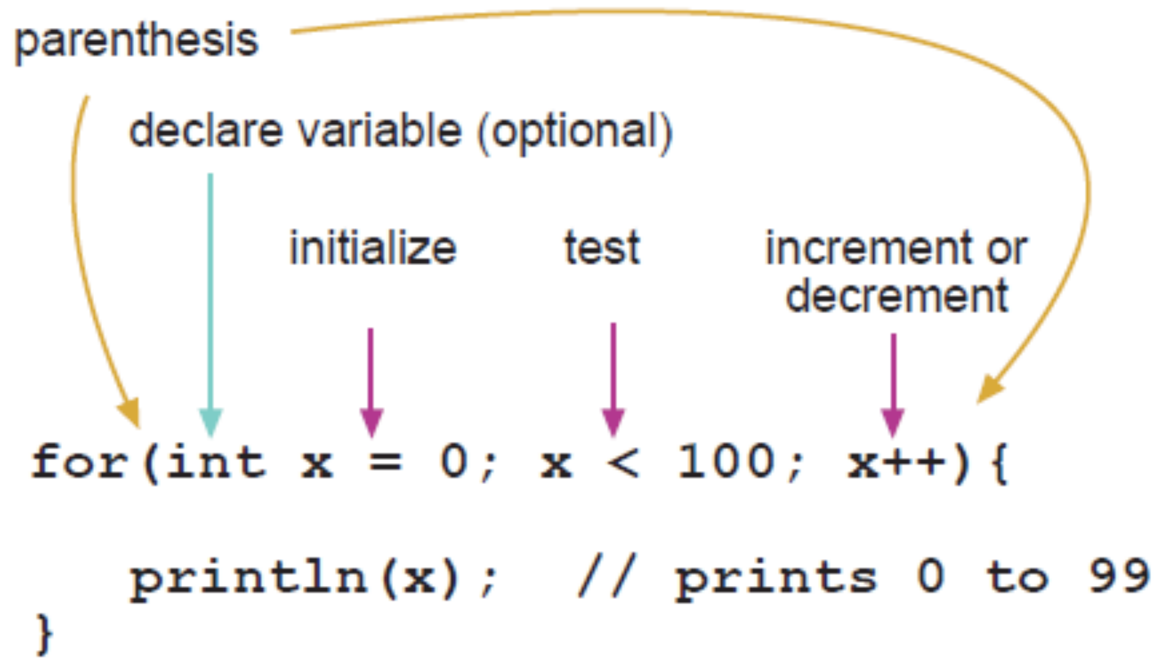
The **for** statement is used to repeat a block of statements enclosed in curly braces.

An increment counter is usually used to increment and terminate the loop.

```
for (initialization; condition; increment) {
   //statement(s);
}
```

for

The **initialization** happens first and exactly once. Each time through the loop, the **condition** is tested; if it's true, the **statement** block, and the **increment** is executed, then the **condition** is tested again. When the **condition** becomes false, the loop ends.



```
for(int x = 0; x < 100; x++){
   println(x);  // prints 0 to 99
}
```

for

Dimming an LED using a PWM pin and a for loop

```
// Dim an LED using a PWM pin

int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10

void setup()
{
  // no setup needed
}

void loop()
{
   for (int i=0; i <= 255; i++){
      analogWrite(PWMpin, i);
      delay(10);
   }
}
```

Functions

`void setup()` is a **function** that is called once, when the program starts.

```
void setup()                        // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);          // sets the digital pin as output
}
```

returned value	**function name**	**(input values)**	**{statements}**
`void`	`setup`	`()`	`{ pinMode(ledPin, OUTPUT); }`

Functions are collections of **statements**. We use **functions** to group together a set of **statements** we want to be able to refer to with one name.

Functions

```
clean_dishes wash_the_dishes(dirty_dishes)
{
  place dirty_dishes in the sink;
  turn on hot water;
  if the water is too hot, turn on cold water too;
  put on some LCD SOUNDSYSTEM to abate boredom;
  add soap to water;
  while dirty_dishes are dirty,
     scrub dirty_dishes until they are clean;
  return clean_dishes to cabinet;
}
```

returned value	**procedure name**	**(input values)**	**{statements}**
clean_dishes	**wash_the_dishes**	(dirty_dishes)	{ ... }

Function calls

`void loop()` is a **function** that is called repeatedly, until the program ends. This is the main function of your program.

```
void loop()                      // run over and over again
{
  digitalWrite(ledPin, HIGH);    // sets the LED on
  delay(1000);                   // waits for a second
  digitalWrite(ledPin, LOW);     // sets the LED off
  delay(1000);                   // waits for a second
}
```

procedure name	**(input value)**	**;**
`delay`	`(1000)`	`;`

This **statement** calls the built-in `delay()` **function**, passing the **input value** of `1000`.

```
void delay(number of milliseconds)
{
  stop and wait for (number of milliseconds);
}
```

Custom functions

You can write your own functions to group a series of statements that you want to execute repeatedly from different places in your code.

First, define your function:

```
void blink()
{
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Then call that function from within your program somewhere:

```
void loop()
{
  blink();
}
```