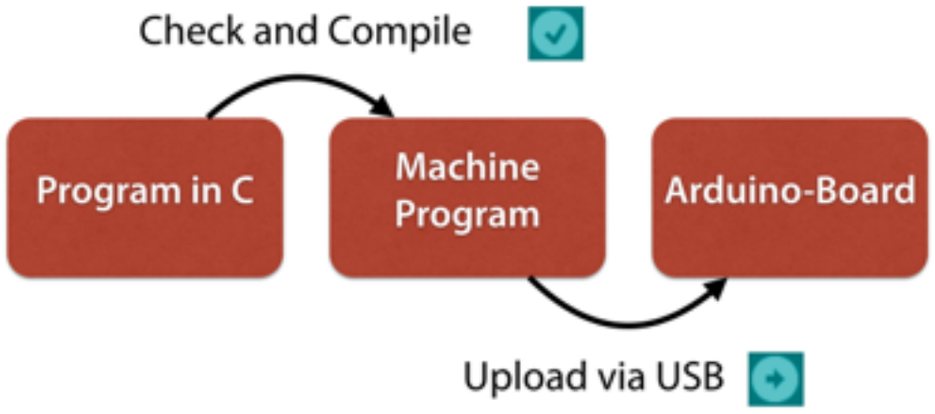
Divide and Conquer

Programming for Engineers
Winter 2015

Andreas Rau, Saarland University

From a Program to a Processor



Function Calls

- Most functions have parameters that determine their mode of operation

```
digitalWrite(pin_number, value)
```

- A value (argument) must be provided for each parameter

```
digitalWrite(13, HIGH);
```

function name — value of pin_number — value of value

Variables

- Variables are used to store values.
- The instruction

```
int led = 13;
```

  introduces led as a variable holding the value 13.
- After this instruction, the value can be accessed via the name led.

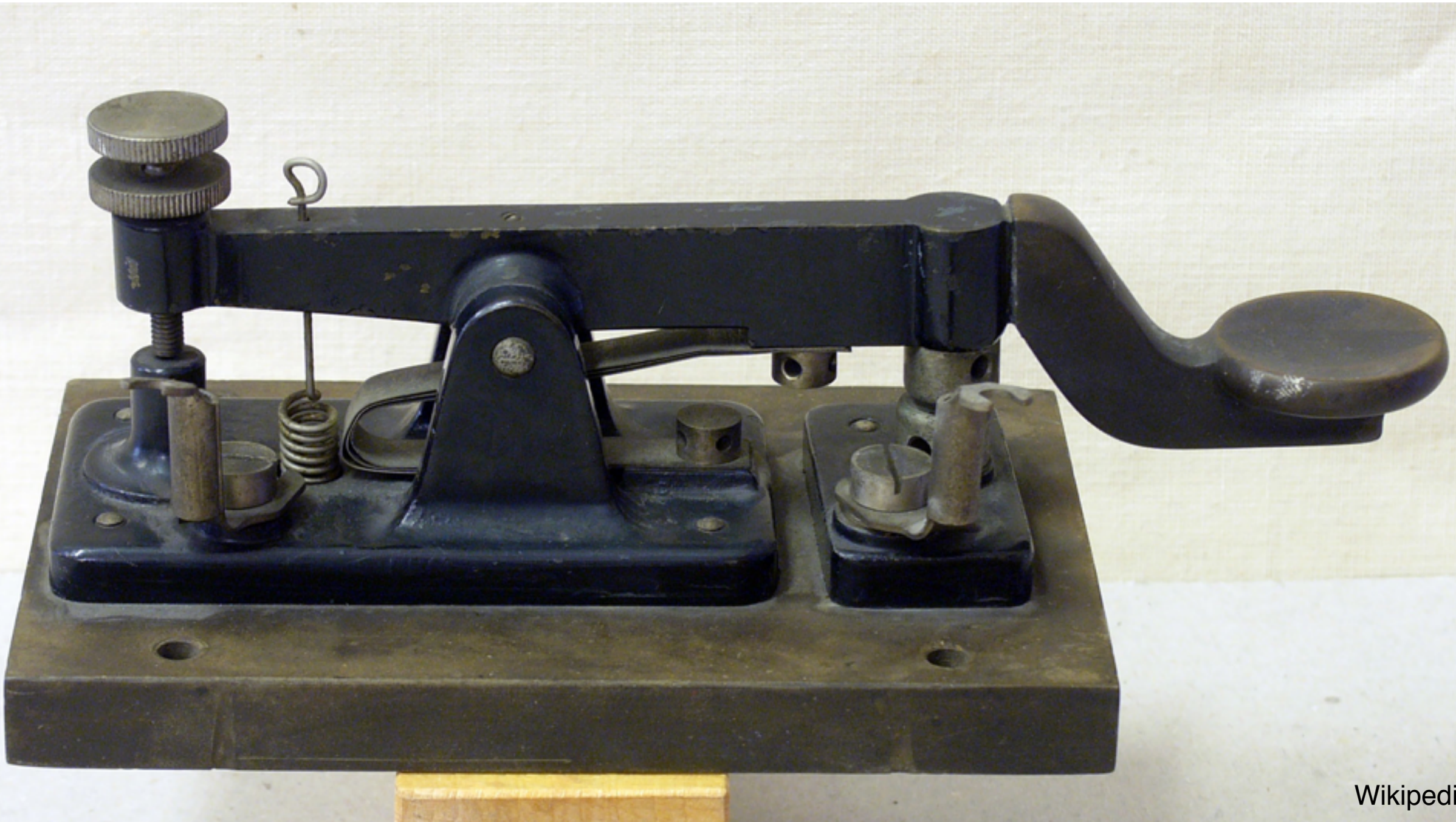
Symbolic Blinking

```
// Pin 13 has an LED connected on most
// Arduino boards.  Give it a name:
int led = 13;

void setup() {
   pinMode(led, OUTPUT);
}

void loop() {
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
}
```

Today's Topics

- Custom functions
- Parameters
- Conditionals
- Debugging

Morse-Code

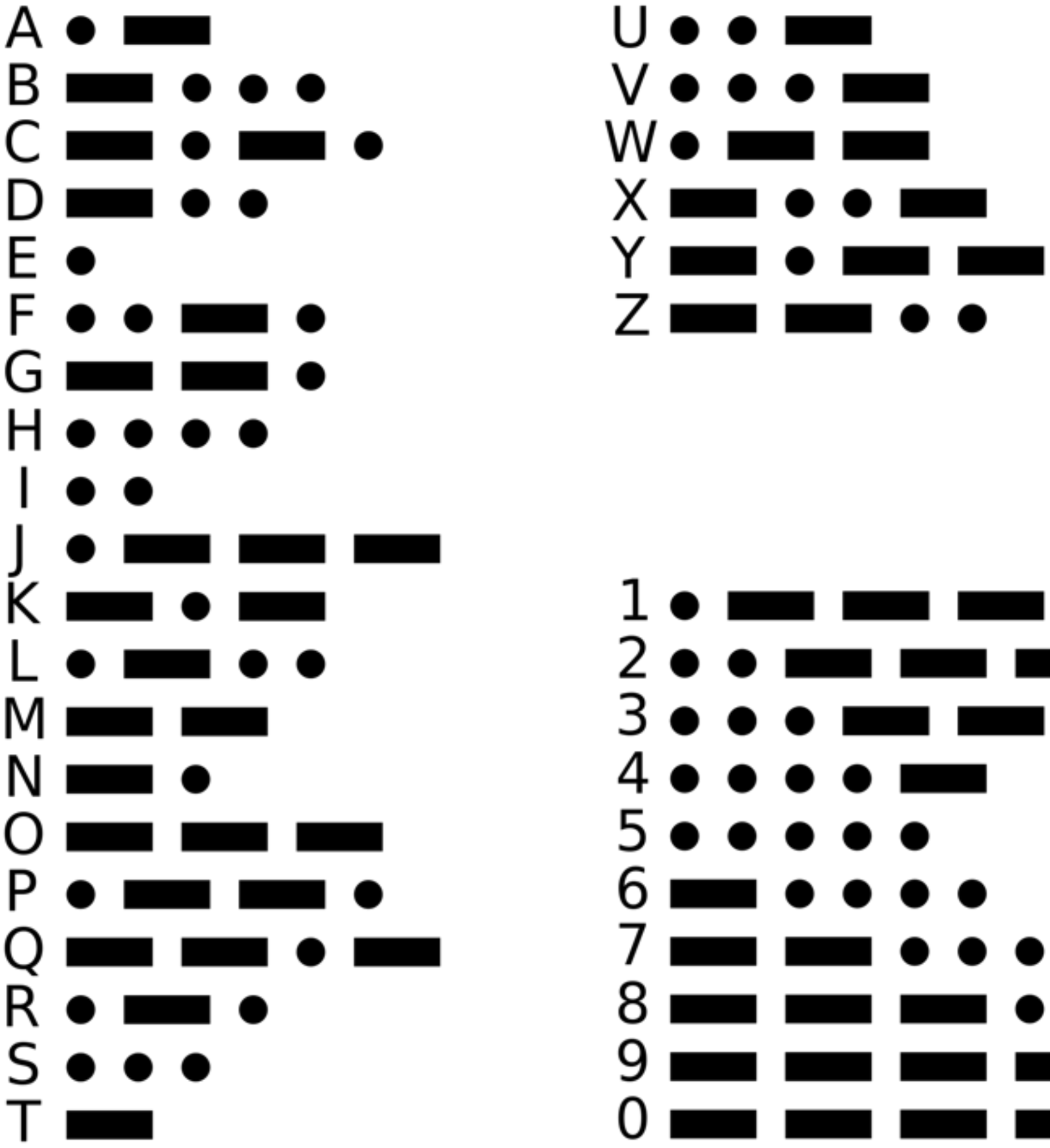


Wikipedia

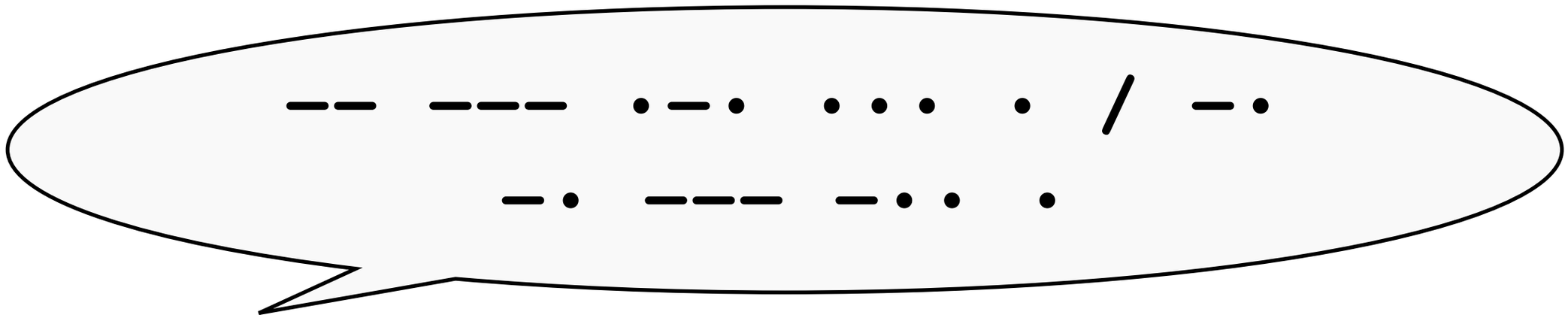
Morse-Code

Consists of three symbols:

- Dot (*Dit*)
- Dash (*Dah*)
- Silence


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
1
2
3
4
5
6
7
8
9
0

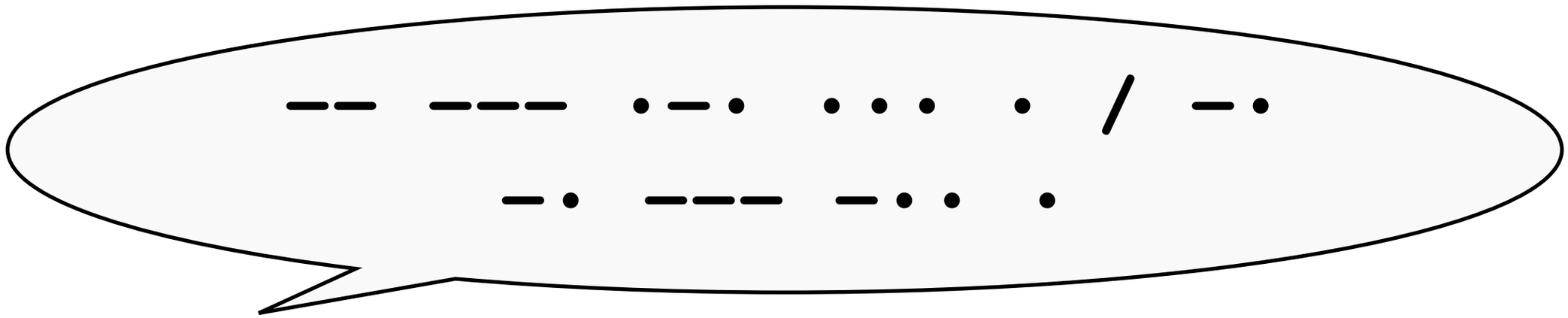
Morse-Code



Morse-Code

dahdah dahdahdah
ditdahdit dididit dit, dahditdahdit dahdahdah
dahditdit dit.

Morse-Code



MORSE CODE

Morse-Code

- A *Dah* is three times as long as a *Dit*.
- The pause between two sent symbols is as long as one *Dit*.
- A pause of the length of a Dah (or three Dits) is inserted between characters in a word.
- There is a pause the length of seven Dits between words.

Wikipedia

```
int dit_delay = 500;                    // length of a dit in ms


void loop() {
    // send a dit
    digitalWrite(led, HIGH);
    delay(dit_delay);
    digitalWrite(led, LOW);
    delay(dit_delay);
}
```

```
int dit_delay = 500;                // length of a dit in ms
int dah_delay = dit_delay * 3; // length of a dah in ms

void loop() {
    // send a dit
    digitalWrite(led, HIGH);
    delay(dit_delay);
```

Computation!

```
    digitalWrite(led, LOW);
    delay(dit_delay);

    // send a dah
    digitalWrite(led, HIGH);
    delay(dah_delay);

    digitalWrite(led, LOW);
    delay(dit_delay);
}
```

Arithmetic Operators

In order of increasing precedence:

1. Addition (+), Subtraction (–)
   Associativity: left to right
2. Multiplication(*), Division(/), Modulo(%)
   Associativity: left to right
3. Algebraic sign (+, –)
   Associativity: right to left

```
int y = -3 + 7 % 3
```

Arithmetic Operators

In order of increasing precedence:

1. Addition (+), Subtraction (–)
   Associativity: left to right

2. Multiplication(*), Division(/), Modulo(%)
   Associativity: left to right

3. Algebraic sign (+, –)
   Associativity: right to left

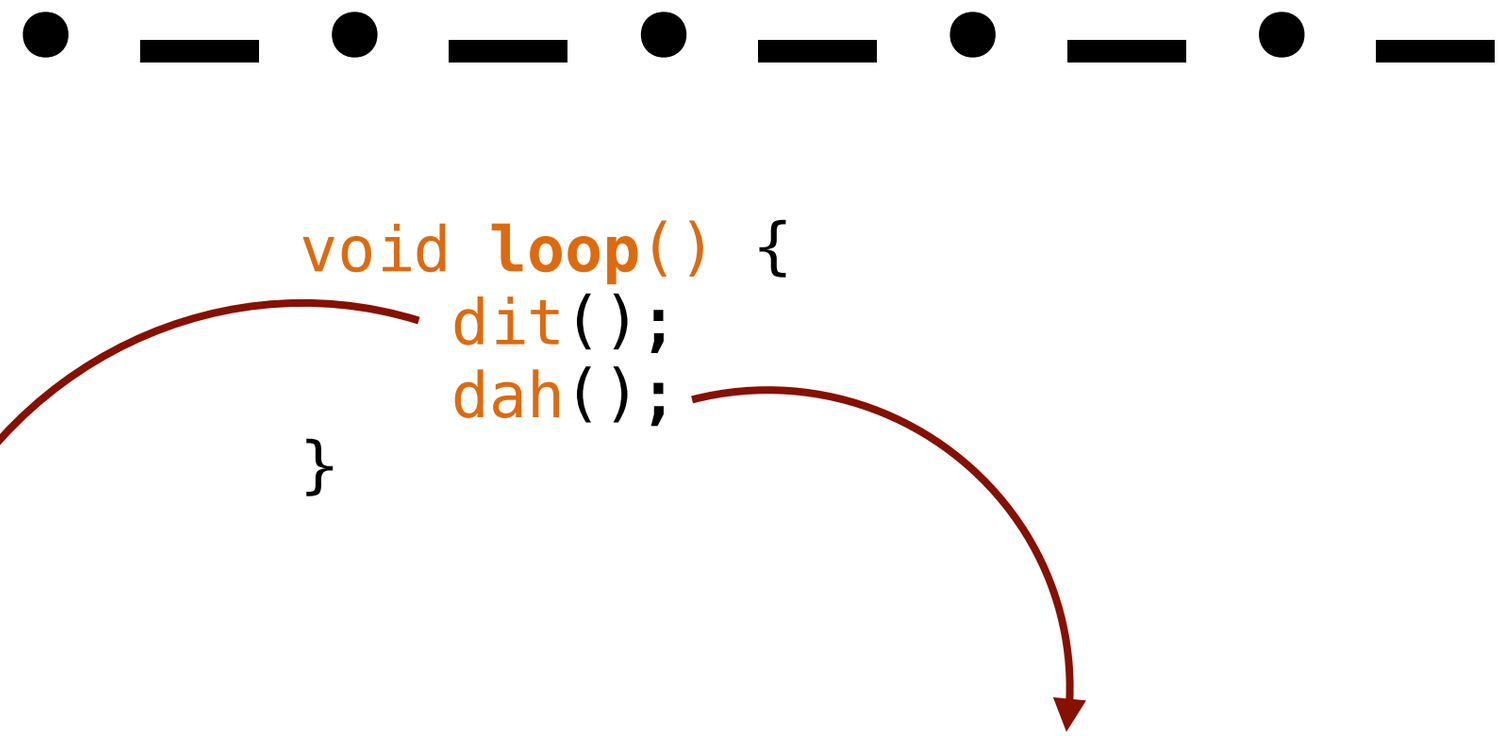
```
int y = -3 + 7 % 3
int y = (-3) + (7 % 3)
```

Custom Functions

- We want to put the instructions for *Dahs* and *Dits* into individual custom *functions*
- A custom function is defined like setup() and loop() as a sequence of instructions:

```
void name() {
    statement 1;
    statement 2;
    …
}
```

```
int dit_delay = 500;              // length of a dit in ms
int dah_delay = dit_delay * 3; // length of a dah in ms

void loop() {
    // send a dit
    digitalWrite(led, HIGH);
    delay(dit_delay);

    digitalWrite(led, LOW);
    delay(dit_delay);

    // send a dah
    digitalWrite(led, HIGH);
    delay(dah_delay);

    digitalWrite(led, LOW);
    delay(dit_delay);
}
```



```
void loop() {
    dit();
    dah();
}
```

```
void dit() {
    // send a dit
    digitalWrite(led, HIGH);
    delay(dit_delay);

    digitalWrite(led, LOW);
    delay(dit_delay);
}
```

```
void dah() {
    // send a dah
    digitalWrite(led, HIGH);
    delay(dah_delay);

    digitalWrite(led, LOW);
    delay(dit_delay);
}
```

Divide and Conquer

- Idea: divide a problem in multiple (smaller) subproblems
- Fundamental principle of computer science
- Fundamental principle of exercising political

Divide et Impera

deditditdit
dedahdahdah

```
int dit_delay = 500;                // length of a dit in ms
int dah_delay = dit_delay * 3; // length of a dah in ms
```

```
void dit() {
    // send a dit
    digitalWrite(led, HIGH);
    delay(dit_delay);

    digitalWrite(led, LOW);
    delay(dit_delay);
}
```

```
void dah() {
    // send a dah
    digitalWrite(led, HIGH);
    delay(dah_delay);

    digitalWrite(led, LOW);
    delay(dit_delay);
}
```

Send an S

```
void morse_S() {
    dit();
    dit();
    dit();
}
```

or (shorter)

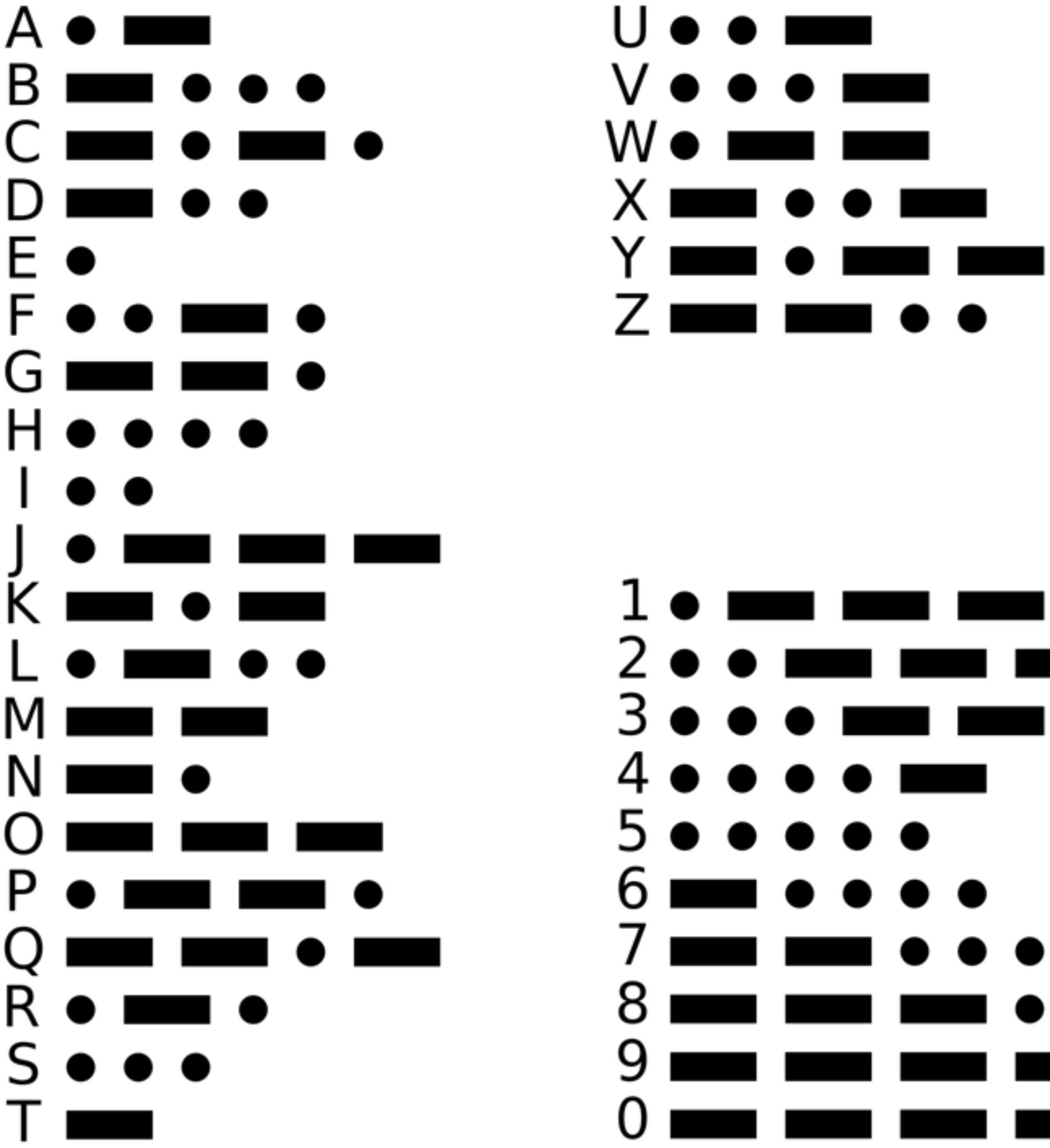
```
void morse_S() {
    dit(); dit(); dit();
}
```

Save Our Souls

```
void morse_S() {
    dit(); dit(); dit();
}
```

```
void morse_O() {
    dah(); dah(); dah();
}

void morse_SOS() {
    morse_S(); morse_O(); morse_S();
    delay(dit_delay * 6);
}
```

• • • — — — • • • / • • • — — — • • • / • • • — — — • • •


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
1
2
3
4
5
6
7
8
9
0

A morse_A()	U morse_U()
B morse_B()	V morse_V()
C morse_C()	W morse_W()
D morse_D()	X morse_X()
E morse_E()	Y morse_Y()
F morse_F()	Z morse_Z()
G morse_G()	
H morse_H()	
I morse_I()	
J morse_J()	
K morse_K()	1 morse_1()
L morse_L()	2 morse_2()
M morse_M()	3 morse_3()
N morse_N()	4 morse_4()
O morse_O()	5 morse_5()
P morse_P()	6 morse_6()
Q morse_Q()	7 morse_7()
R morse_R()	8 morse_8()
S morse_S()	9 morse_9()
T morse_T()	0 morse_0()

SINK

```
void morse_S() {
    dit(); dit(); dit();
}
```

```
void morse_I() {
    dit(); dit();
}
```

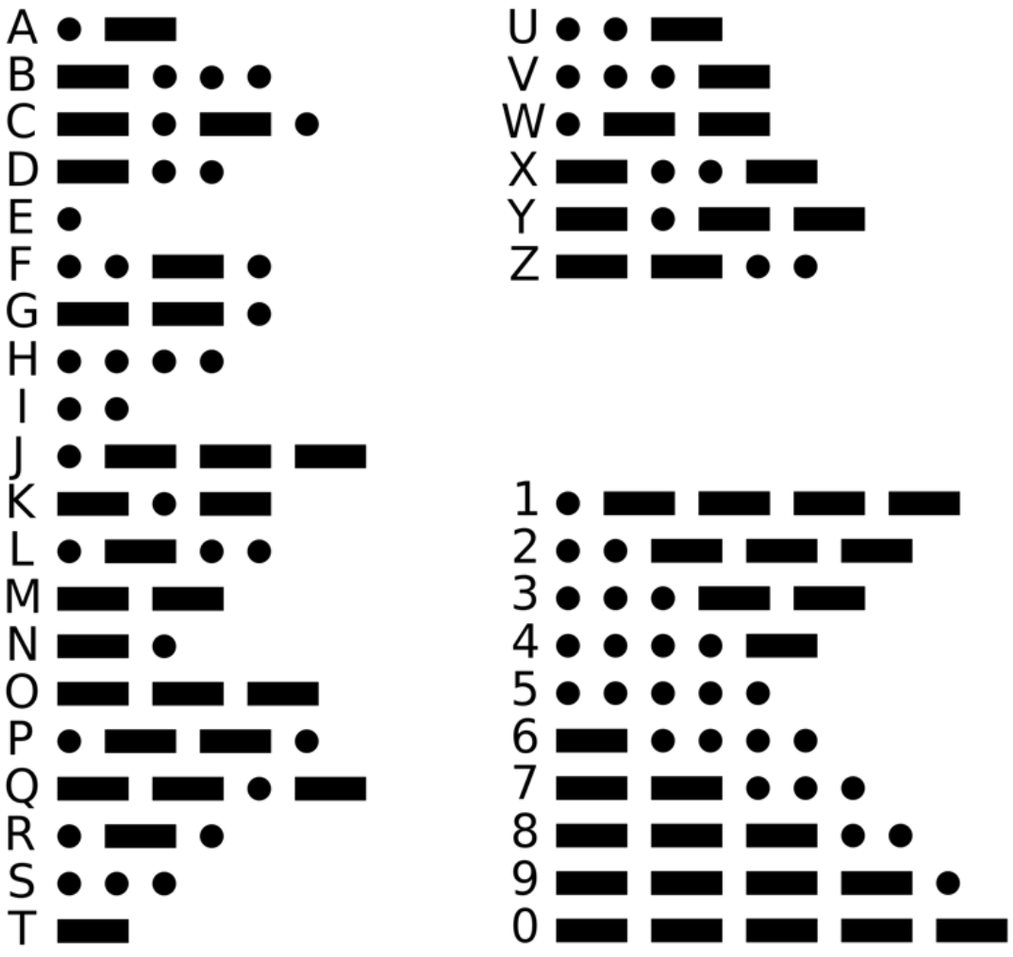
SINK

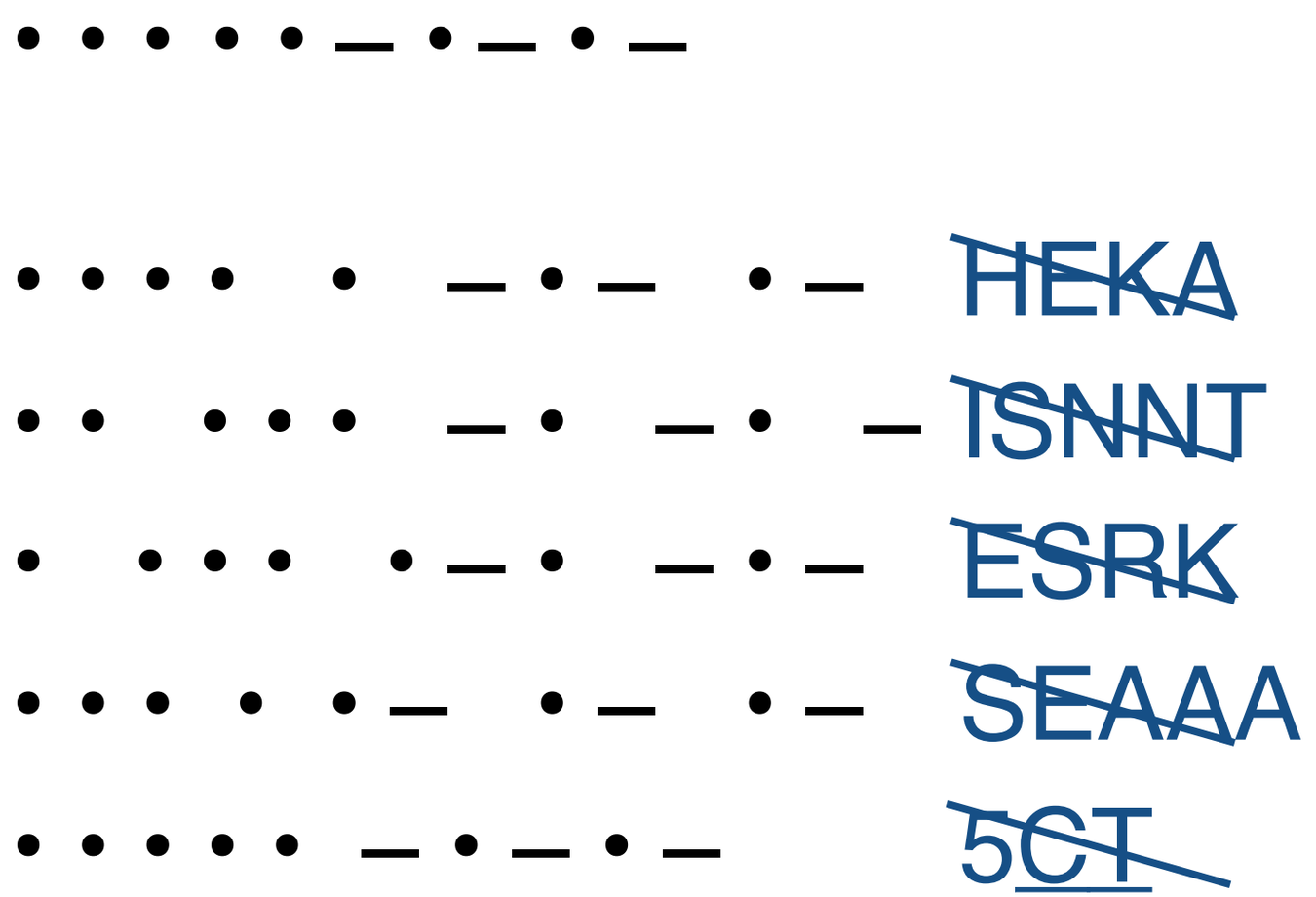
```
void morse_S() {
    dit(); dit(); dit();
}
```

```
void morse_I() {
    dit(); dit();
}
```

```
void morse_SINK() {
    morse_S(); morse_I(); morse_N(); morse_K();
     • • •        • •         — •         — • —
}
```

A	●▬	U	●●▬
B	▬●●●	V	●●●▬
C	▬●▬●	W	●▬▬
D	▬●●	X	▬●●▬
E	●	Y	▬●▬▬
F	●●▬●	Z	▬▬●●
G	▬▬●		
H	●●●●		
I	●●		
J	●▬▬▬		
K	▬●▬	1	●▬▬▬▬
L	●▬●●	2	●●▬▬▬
M	▬▬	3	●●●▬▬
N	▬●	4	●●●●▬
O	▬▬▬	5	●●●●●
P	●▬▬●	6	▬●●●●
Q	▬▬●▬	7	▬▬●●●
R	●▬●	8	▬▬▬●●
S	●●●	9	▬▬▬▬●
T	▬	0	▬▬▬▬▬


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
1
2
3
4
5
6
7
8
9
0


HEKA
TSNNT
ESRK
SEAAA
5CT

Si tacuisses

```
void morse_S() {
    dit(); dit(); dit();
}
```

```
void morse_I() {
    dit(); dit();
}
```

```
void morse_SINK() {
    morse_S(); morse_I(); morse_N(); morse_K();
}
```

• • • • • — • — • —

Si tacuisses

```
void morse_S() {
    dit(); dit(); dit();
    pause_letter();
}

void morse_I() {
    dit(); dit();
    pause_letter();
}

void morse_SINK() {
    morse_S(); morse_I(); morse_N(); morse_K();
    pause_word();
}
```

• • • • • — • — • —

Si tacuisses

```
int dit_delay = 500;                // length of a dit in ms
int dah_delay = dit_delay * 3; // length of a dah in ms


// dit() and dat() already include dit_delay
int letter_delay = dah_delay - dit_delay;

// letters already include letter delay
int word_delay   = dit_delay * 7 - letter_delay;


void pause_letter() {
   delay(letter_delay);
}
```

```
void pause_word() {
   delay(word_delay);
}
```

• • • • • — • — • — SINK

A • —
B — • • •
C — • — •
D — • •
E •
F • • — •
G — — •
H • • • •
I • •
J • — — —
K — • —
L • — • •
M — —
N — •
O — — —
P • — — •
Q — — • —
R • — •
S • • •
T —

U • • —
V • • • —
W • — —
X — • • —
Y — • — —
Z — — • •

1 • — — — —
2 • • — — —
3 • • • — —
4 • • • • —
5 • • • • •
6 — • • • •
7 — — • • •
8 — — — • •
9 — — — — •
0 — — — — —

Custom Parameters

- Goal: write a function send_number(*n*), that outputs the morse code for the number n
- *n* shall be the parameter of the function

Custom Parameters

- Parameters (along with their types) are declared in parentheses

```
void name(int p1, int p2, ...) {
    Instructions…;
}
```

- In our case:

```
void morse_number(int n) {
    Instructions…;
}
```

Conditionals

- Different instructions must be executed depending on the value of n:
  - if $n = 1$, then send • – – – –
  - if $n = 2$, then send • • – – –
  - etc.

Conditionals

- The if-clause enables to express conditionals

```
if (condition) {
    Instructions…;
}
```

- The instructions are only executed if the condition holds

Comparison Operators

In order of increasing precedence:

1. Logical Or ∨ (|| |)
2. Logical And ∧ (&&)
3. Comparators (<, >, <=, >=)
4. Equality = (==), Inequality ≠ (!=)
5. Logical Not ¬ (!)

==, not = !

```
if (x >= y && !(x == y))
```

Conditionals

- Different instructions must be executed depending on the value of n:
  - if $n = 1$, then send • – – – –
  - if $n = 2$, then send • • – – –
  - etc.

Conditionals

```
// send n in morse code
void morse_digit(int n) {
    if (n == 0) {
        dah(); dah(); dah(); dah(); dah();
    }
    if (n == 1) {
        dit(); dah(); dah(); dah(); dah();
    }
```

```
// send n in morse code
void morse_digit(int n) {
    if (n == 0) {
        dah(); dah(); dah(); dah(); dah();
    }
    if (n == 1) {
        dit(); dah(); dah(); dah(); dah();
    }
    if (n == 2) {
        dit(); dit(); dah(); dah(); dah();
    }
    // etc. for 3–8
    if (n == 9) {
        dah(); dah(); dah(); dah(); dit();
    }
    pause_letter();
}
```

Function Call

- Once defined, morse_digit() can be called like any other function:

```
void morse_digit(int n) {
    // as above
}

void loop() {
    morse_digit(5);
    morse_digit(0);
    morse_digit(2);
    morse_digit(4);
}
```

From Digits to Numbers

- How do we send out multi-digit numbers?
- Goal: A function morse_number(*n*), that

```
morse_number(5024) →
    morse_digit(5)
    morse_digit(0)
    morse_digit(2)
    morse_digit(4)
```

From Digits to Numbers

- Observation: if I want to send out 5024, I can send out 502 followed by 4.

```
morse_number(5024) →
    morse_number(502)
    morse_digit(4)
```

From Digits to Numbers

- Observation: if I want to send out 5024, I can send out 502 followed by 4.

```
morse_number(5024) →
    morse_number(502)
    morse_digit(4)
```

- To send out 502 I can send out 50 followed by 2.

```
morse_number(502) →
    morse_number(50)
    morse_digit(2)
```

From Digits to Numbers

Common principle:

1. If n has more than one digit (i.e. $n \geq 10$), send out n / 10 first
2. Afterwards send out the last digit (i.e. n mod 10)

From Digits to Numbers

morse_number() looks like this:

```
void morse_number(int n) {
    if (n >= 10) {
        morse_number(n / 10);
    }
    morse_digit(n % 10);
}
```

From Digits to Numbers

```
void morse_number(int n) {
    if (n >= 10) {
        morse_number(n / 10);
    }
    morse_digit(n % 10);
}
```

```
morse_number(5024)
    → morse_number(502)
        → morse_number(50)
            → morse_number(5)
                → morse_digit(5)    • • • • •
            → morse_digit(0)        — — — — —
        → morse_digit(2)            • • — — —
    → morse_digit(4)                • • • • —
```

Recursion

- Recursion is when a function makes a call to itself
- Every computation can be expressed using only *functions*, *conditionals and recursion*
- You are now able to program everything that is (somehow) computable

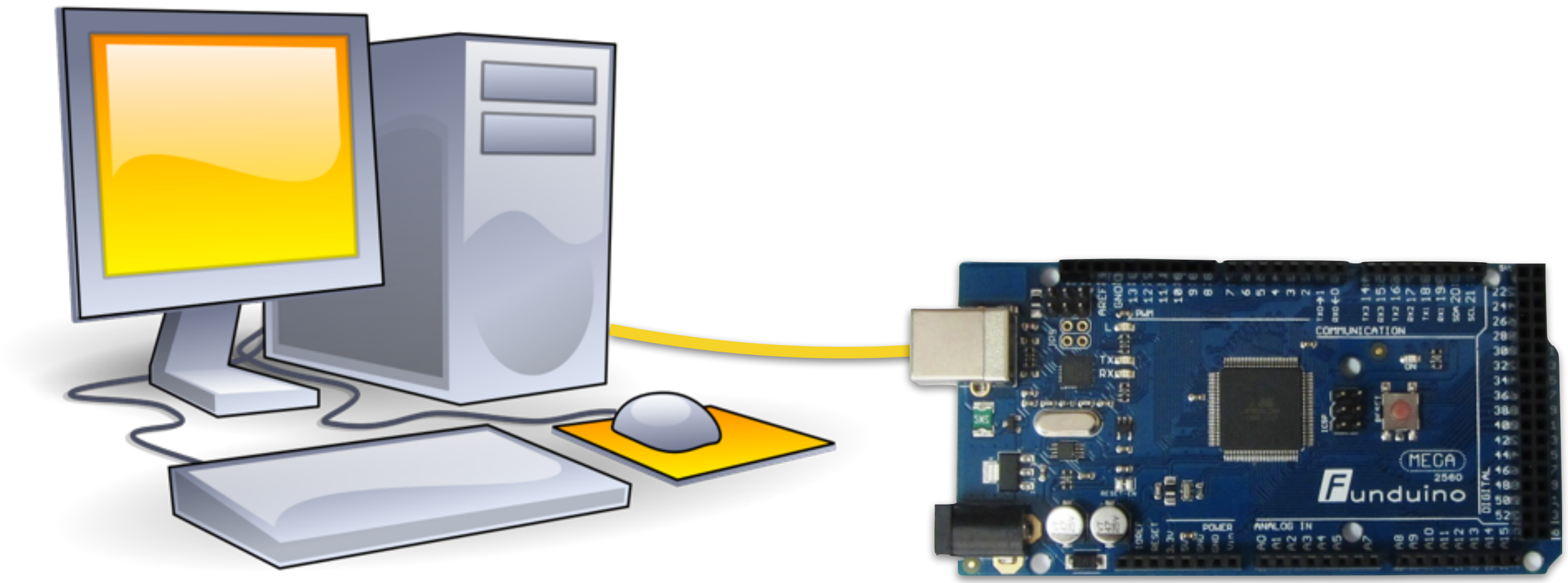
(at least in principle)

Debugging

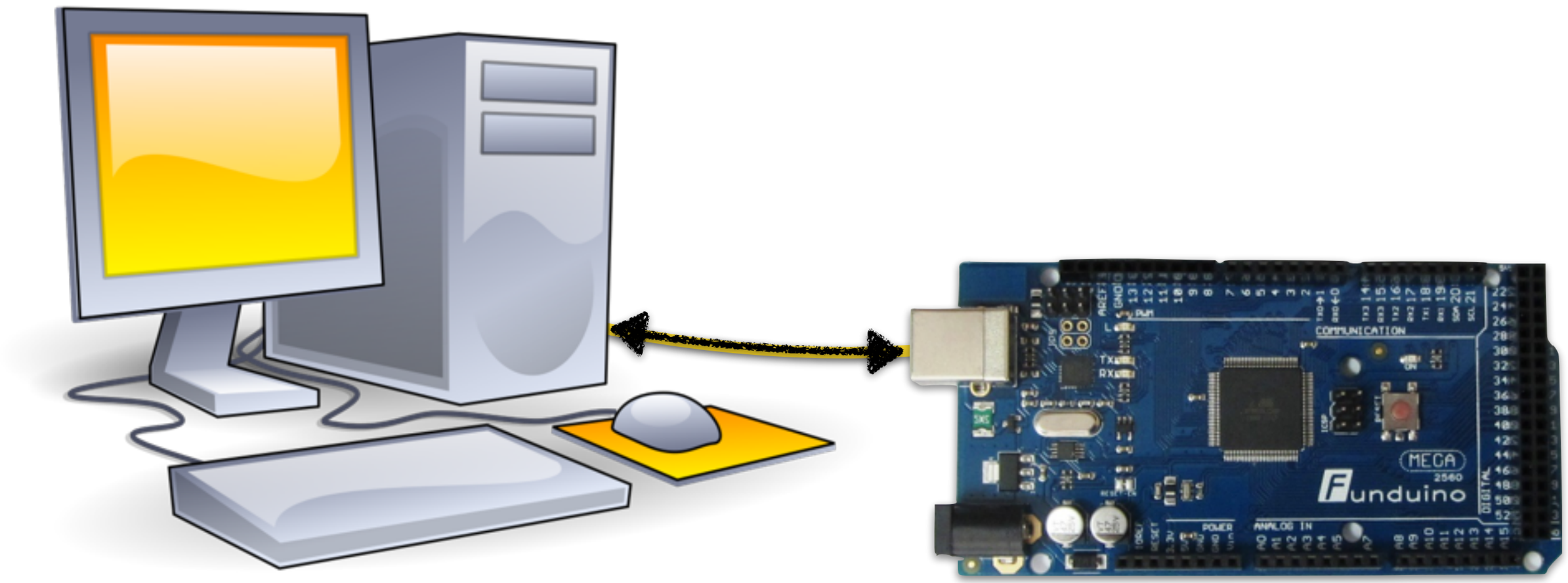
Debugging

- It is often helpful to follow a computation closely as it happens
- The serial interface of the Arduino platform makes this possible

USB-Connection



Data Transfer



Monitoring with tools → serial monitor

Serial.begin()

- Serial.begin(*baud*) sets up the serial interface to transfer data with speed *baud* (bits/s)

- Example:

```
void setup() {
    // Transfer at 9600 bits/s
    Serial.begin(9600);
}
```

Serial.print()

- The function Serial.print(*x*) prints *x* on the serial interface
- Serial.println(x): Similarly, but with a line end
- Example:

```
void morse_number(int n) {
    Serial.println(n);
    Instructions…;
}
```

Printing Text

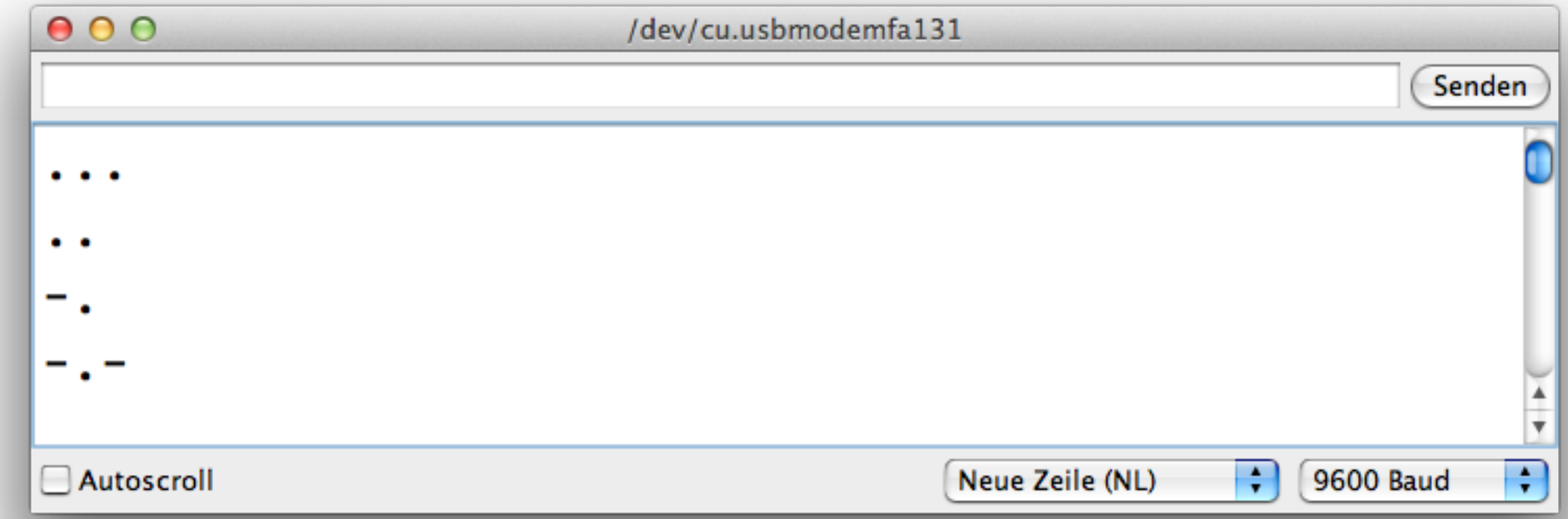
- Serial.print() and Serial.println() can also be used to print out text
- Text is enclosed within "…"
- Example:

```
void morse_number(int n) {
    Serial.print("morse_number(");
    Serial.print(n);
    Serial.println(")");
    Instructions…;
}
```

Printing Text

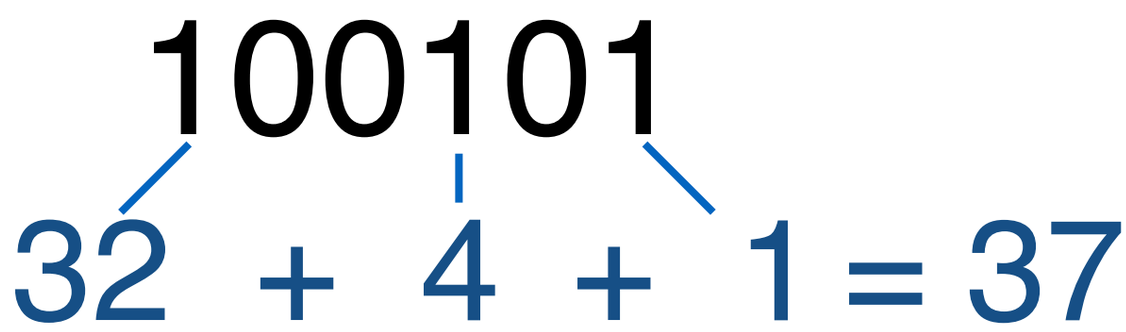
```
void dit() {
    Serial.print(".");
    Instructions…;
}

void dah() {
    Serial.print("-");
    Instructions…;
}

void pause_letter() {
    Serial.println("");
    delay(letter_delay);
}
```

Serial Monitor



Binary Numbers

- Computers represent numbers internally using bits – only 0 and 1
- Numbers are stored in the binary system
- For example, the number 37 is stored



From Digits to Numbers

morse_number() looks like this:

```
void morse_number(int n) {
    if (n >= 10) {
        morse_number(n / 10);
    }
    morse_digit(n % 10);
}
```

Can we also use a different base than 10?

Binary Number Morse

Printing binary numbers in morse code:

```
void morse_binary(int n) {
    if (n >= 2) {
        morse_binary(n / 2);
    }
    morse_digit(n % 2);
}
```

Base 10 and 2

```
void morse_decimal(int n) {
    if (n >= 10) {
        morse_decimal(n / 10);
    }
    morse_digit(n % 10);
}

void morse_binary(int n) {
    if (n >= 2) {
        morse_binary(n / 2);
    }
    morse_digit(n % 2);
}
```

Arbitrary Base

Printing numbers in base:

```
void morse_base(int n, int base) {
     if (n >= base) {
          morse_base(n / base, base);
     }
     morse_digit(n % base);
}
```

Outlook

- Assignments
- Custom loops
- Traffic control
- Input elements

Custom Functions

```
void loop() {
    dit();
    dah();
}
```

```
void dit() {
    // send a dit
    digitalWrite(led, HIGH);
    delay(dit_delay);

    digitalWrite(led, LOW);
    delay(dit_delay);
}
```

```
void dah() {
    // send a dah
    digitalWrite(led, HIGH);
    delay(dah_delay);

    digitalWrite(led, LOW);
    delay(dit_delay);
}
```

Custom Parameters

- Parameters (along with their types) are declared in parentheses

```
void name(int p1, int p2, ...) {
    Instructions…;
}
```

- In our case:

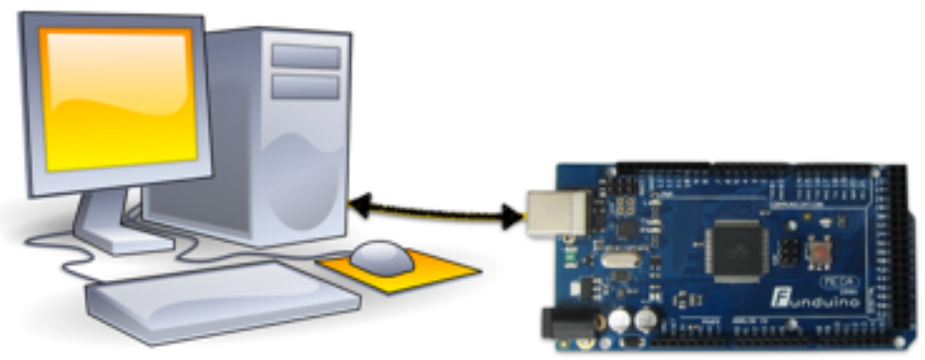
```
void morse_number(int n) {
    Instructions…;
}
```

Recursion

```
void morse_number(int n) {
    if (n >= 10) {
        morse_number(n / 10);
    }
    morse_digit(n % 10);
}
```

```
morse_number(5024)
    → morse_number(502)
        → morse_number(50)
            → morse_number(5)
                → morse_digit(5)    • • • • •
            → morse_digit(0)        – – – – –
        → morse_digit(2)            • • – – –
    → morse_digit(4)                • • • • –
```

Monitoring the Process



Monitoring with tools → serial monitor