Λογισμικό των Συστημάτων Ανάπτυξης Arduino

Ψηφία των διαφόρων αριθμητικών συστημάτων

Δεκαδικό	Δυαδικό	Οκταδικό	Δεκαεξαδικό
0	0	0	0
1	1	1	1
2		2	2
3		3	3
4		4	4
5		5	5
6		6	6
7		7	7
8			8
9			9
			A
			B
			C
			D
			E
			F

Δυνάμεις του 2

n	2^n	n	2^n
0	1	8	256
1	2	9	512
2	4	10	1024
3	8	11	2048
4	16	12	4096
5	32	13	8192
6	64	14	16384
7	128	15	32768

$2^{-1}=0.5$

$2^{-2}=0.25$

$2^{-3}=0.125$

$2^{-4}=0.0625$

$2^{-5}=0.03125$

Μετατροπή του αριθμού 125_{10} του δεκαδικού συστήματος στο δυαδικό.

125 = 2x62+1 → 1 LSB
62 = 2x31+0 → 0
31 = 2x15+1 → 1
15 = 2x7+1 → 1
7 = 2x3+1 → 1
3 = 2x1+1 → 1
1 = 2x0+1 → 1 MSB

Άρα $125_{10} = 1111101_2$

Να μετατραπεί ο αριθμός $N = 10011$ του δυαδικού συστήματος αρίθμησης στο δεκαδικό σύστημα.

Λύση

$$N = 1\times2^4+0\times2^3+0\times2^2+1\times21+1\times2^0$$
$$= 1\times16+0\times4+1\times2+1\times1$$
$$= 19$$

Παράσταση προσημασμένων αριθμών με 4 bit σε συμπλήρωμα του 2

Δεκαδικός	Παράσταση σε Συμπλήρωμα του 2
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

Κώδικας ASCII

Σε πολλά ψηφιακά συστήματα απαιτείται η αναπαράσταση εκτός των αριθμών και άλλων χαρακτήρων, όπως γραμμάτων, σημείων στίξης, συμβόλων, κλπ. Για τον σκοπό αυτό δημιουργήθηκαν οι αλφαριθμητικοί κώδικες, ο πιο διαδεδομένος από τους οποίους είναι ο κώδικας ASCII (American Standard Code for Information Interchange). Ο *κώδικας ASCII* χρησιμοποιεί 7 bit για την κωδικοποίηση 128 (=2^7) χαρακτήρων. Ο κώδικας ASCII ξεκίνησε από τις ΗΠΑ και σύντομα τυποποιήθηκε διεθνώς σαν IA5 (International Alphabet 5). Από τους 128 χαρακτήρες οι 95 είναι χαρακτήρες γραφής και οι 33 χαρακτήρες ελέγχου. Οι χαρακτήρες ελέγχου χρησιμοποιούνται για να ενεργοποιήσουν ή να τροποποιήσουν λειτουργίες κατά την εγγραφή, την επεξεργασία και την μετάδοση δεδομένων.

Κώδικας ASCII

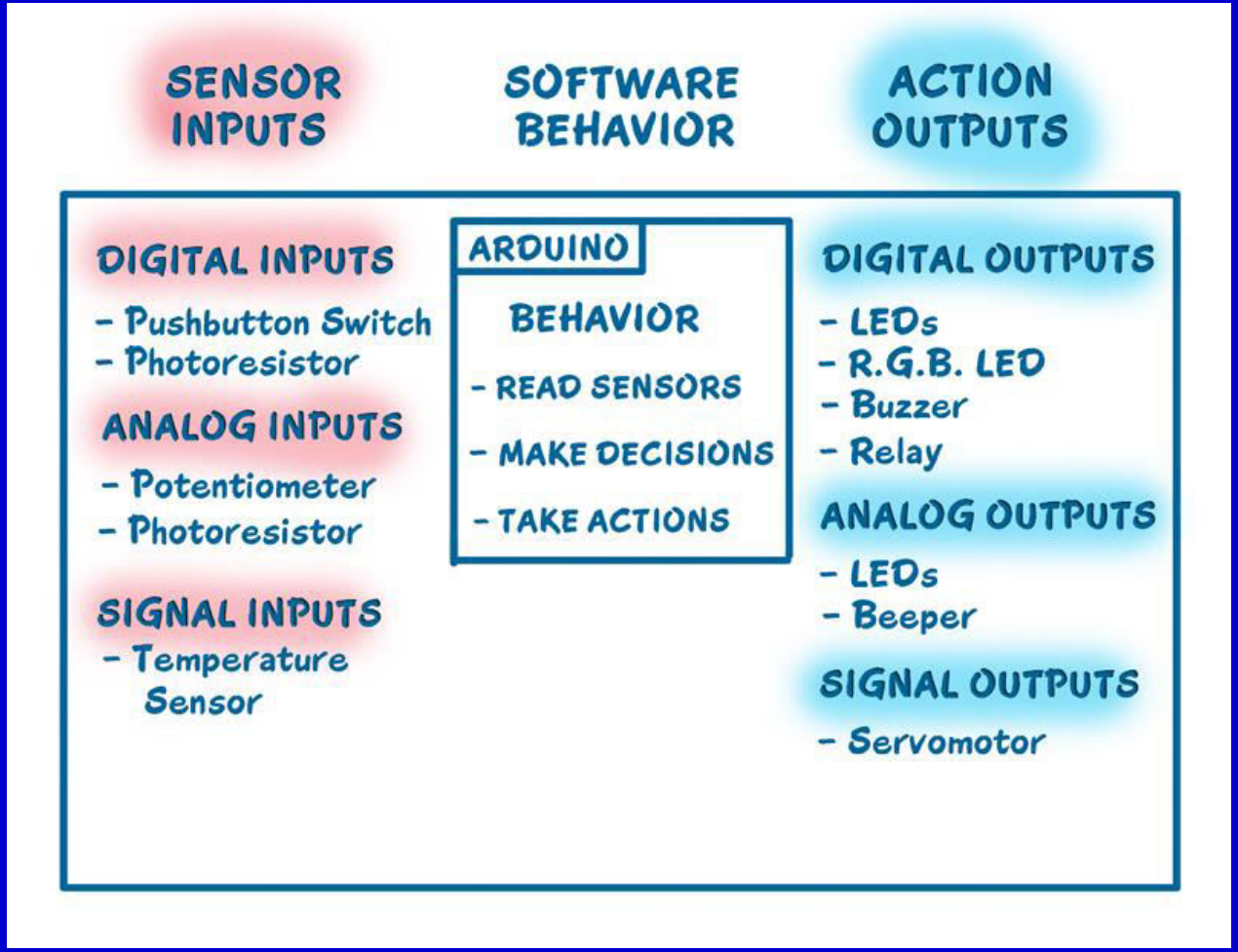
	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Unicode και UTF-8

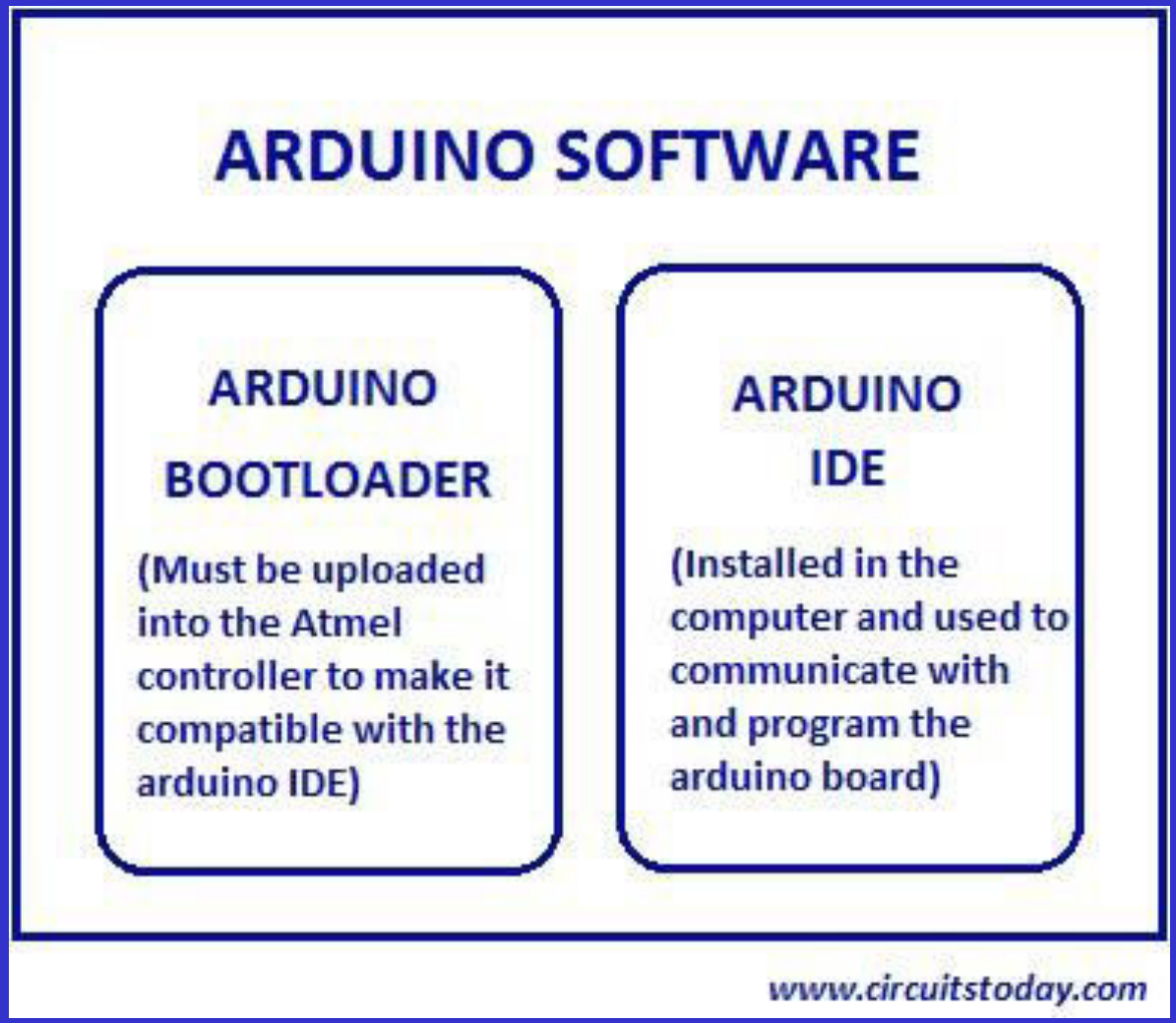
Ο *Unicode* είναι ένα πρότυπο για την αποδοτική κωδικοποίηση, αναπαράσταση και την επεξεργασία κειμένου γραμμένου στα περισσότερα συστήματα γραφής που υπάρχουν στον κόσμο. Ο Unicode υλοποιείται με διαφορετικούς τρόπους. Οι ποιο διαδεδομένες υλοποιήσεις του Unicode είναι οι UTF-8, UTF-16 και η εκλείπουσα UCS-2.

Ο κώδικας UTF-8 (Unicode Transformation Format-8) είναι μία παραλλαγή του Unicode με μεταβλητού μήκους κωδικοποίηση των χαρακτήρων. Ο UTC-8 χρησιμοποιεί ένα byte για τους χαρακτήρες του ASCII, δύο byte για την κωδικοποίηση των περισσοτέρων Ευρωπαϊκών γλωσσών και τρία ή τέσσερα byte για την κωδικοποίηση χαρακτήρων αλλά και ιδεογραμμάτων ασιατικών γλωσσών.

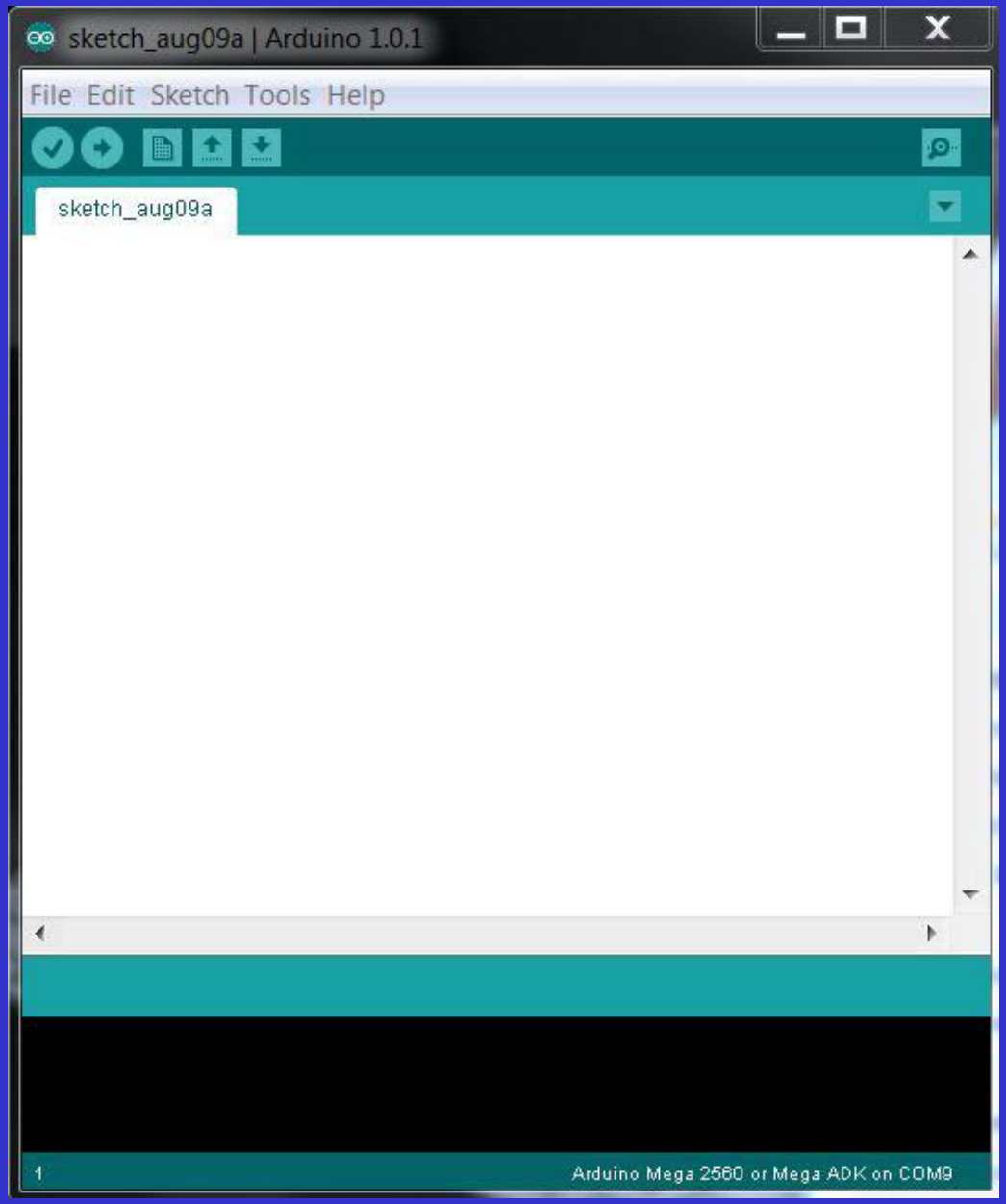
Δομή εφαρμογής με Arduino



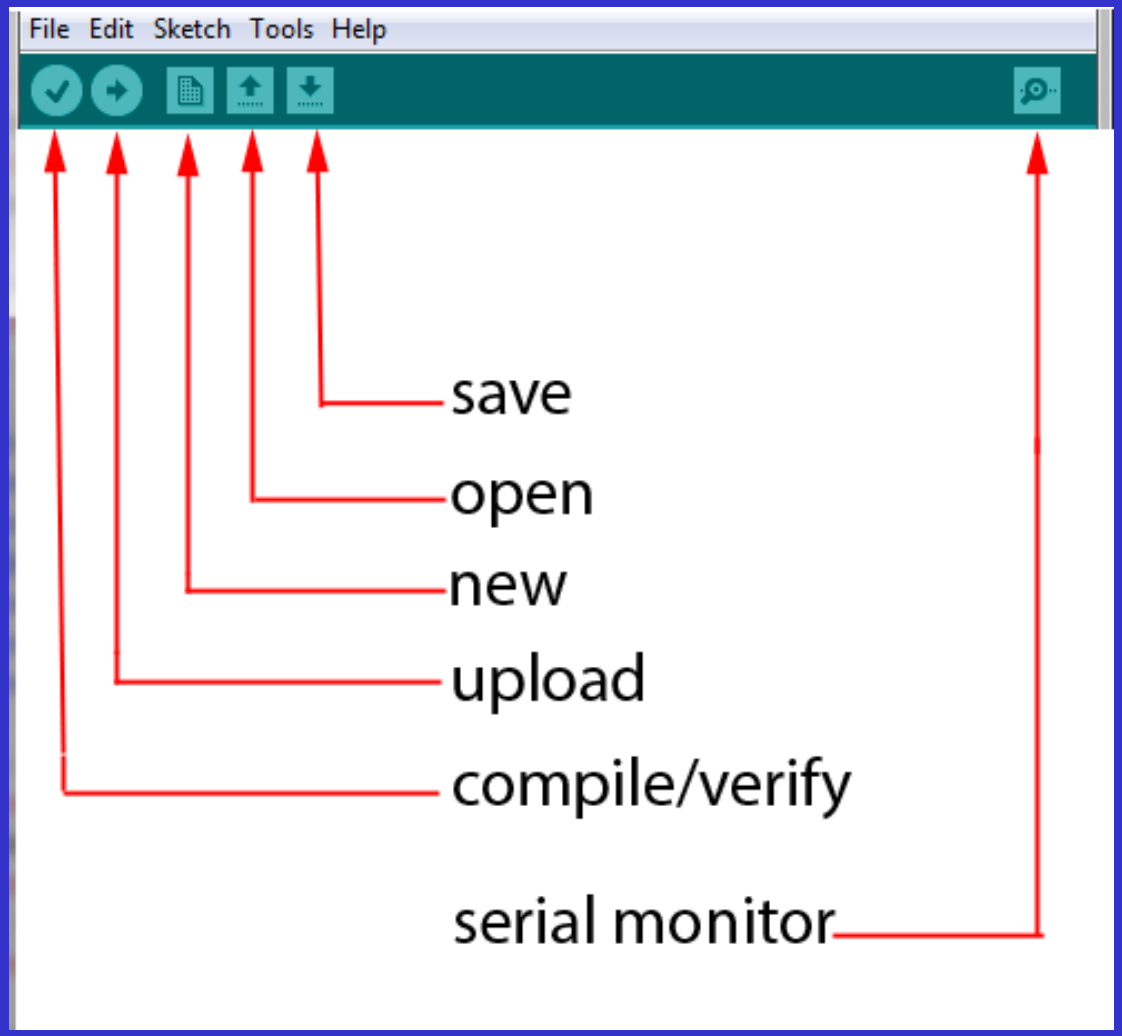
Σύνδεση PC με Arduino


ARDUINO SOFTWARE
ARDUINO BOOTLOADER
(Must be uploaded into the Atmel controller to make it compatible with the arduino IDE)
ARDUINO IDE
(Installed in the computer and used to communicate with and program the arduino board)
www.circuitstoday.com

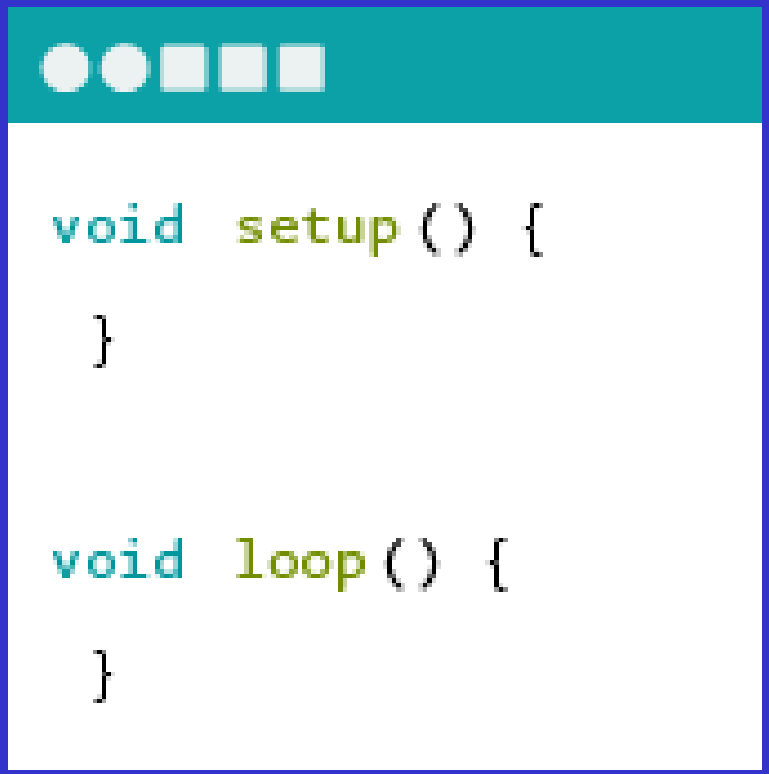
Arduino IDE


sketch_aug09a | Arduino 1.0.1
File Edit Sketch Tools Help
sketch_aug09a
1
Arduino Mega 2560 or Mega ADK on COM9

Arduino IDE


File Edit Sketch Tools Help
save
open
new
upload
compile/verify
serial monitor

Δομή προγραμμάτων του Arduino



```
void setup() {
}

void loop() {
}
```

Δομή προγραμμάτων του Arduino

- `void setup()`
  - Will be executed only when the program begins (or reset button is pressed)
- `void loop()`
  - Will be executed repeatedly

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

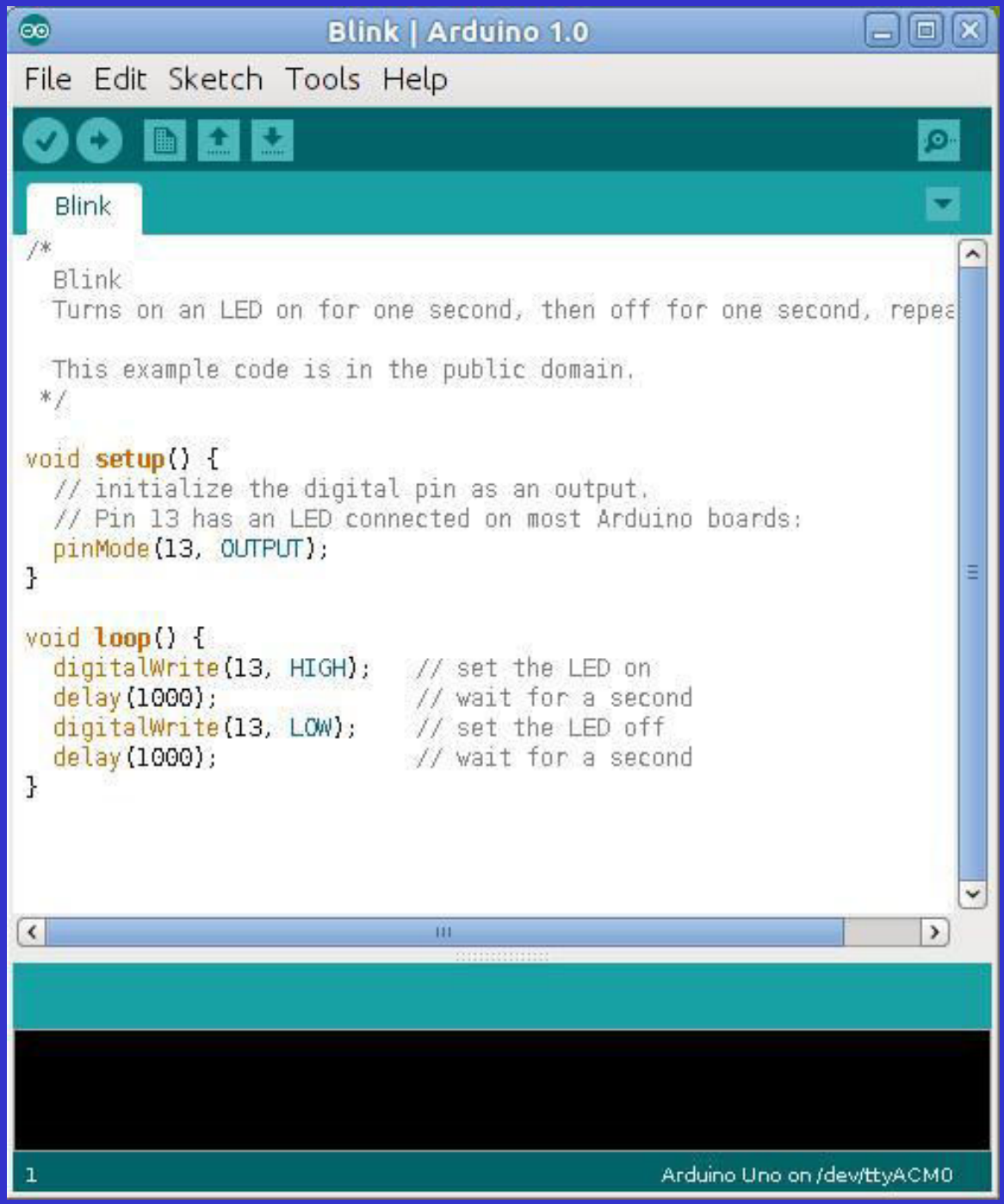
Text that follows // is a comment (ignored by compiler)

Useful IDE Shortcut: Press Ctrl-/ to comment (or uncomment) a selected portion of your program.

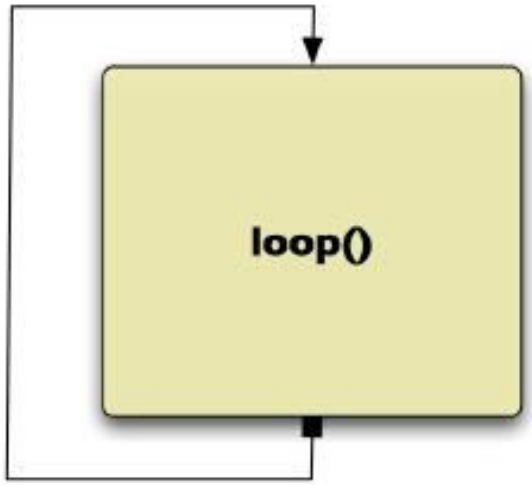
The **setup() function** is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.

The **loop() function** does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Παράδειγμα προγράμματος για Arduino



An Arduino "Sketch"



- Declare variables at top
- Initialize
  - setup() – run once at beginning, set pins
- Running
  - loop() – run repeatedly, after setup()

Constants

Constants are predefined expressions in the Arduino language. They are used to make the programs easier to read.

true | false constants

false is defined as 0 (zero).
true is often said to be defined as 1, which is correct, but true has a wider definition. Any integer which is non-zero is true, in a Boolean sense. So -1, 2 and -200 are all defined as true, too, in a Boolean sense.

integer constants

Integer *constants* are numbers used directly in a sketch, like 123.

Base	Example	Formatter	Comment
10 (decimal)	123	none	
2 (binary)	B1111011	leading 'B'	only works with 8 bit values (0 to 255) characters 0-1 valid
8 (octal)	0173	leading "0"	characters 0-7 valid
16 (hexadecimal)	0x7B	leading "0x"	characters 0-9, A-F valid

Floating point constants

Similar to integer constants, ***floating point constants*** are used to make code more readable. Floating point constants are swapped at compile time for the value to which the expression evaluates.

floating-point constant	evaluates to:	also evaluates to:
10.0	10	
2.34E5	2.34 * 10^5	234000
67e-12	67.0 * 10^-12	.000000000067

Arduino data types

void
bool
boolean
char
unsigned char
byte
int
unsigned int
word
long
unsigned long
short
float
double
string-char array
string-object array

void

The ***void*** keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

bool

A ***bool*** holds one of two values, true or false. (Each bool variable occupies one byte of memory.)

Boolean

boolean is a non-standard type alias for bool defined by Arduino. It's recommended to instead use the standard type bool, which is identical.

char

A *char variable* takes up **1-byte** of memory that stores a character value.
Character literals are written in single quotes, like this: 'A' (for multiple characters-strings-use double quotes: "ABC").
The char data type is a signed type, meaning that it encodes numbers from -128 to +127.

```
char myChar = 'A';
char myChar = 65;
```

unsigned char

unsigned char is an unsigned data type that occupies 1 byte of memory. It is same as the byte data type. The unsigned char data type encodes numbers from 0 to 255.
For consistency of Arduino programming style, **the *byte*** data type is to be preferred.

```
unsigned char myChar = 240;
```

byte

A ***byte*** data type stores an **8-bit** unsigned number, from 0 to 255.

```
byte b = B10010; // "B" is the binary formatter
                    (B10010=18 decimal)
```

int

int (integer) is the primary data type for number storage.

On the Arduino Uno (and other ATMega based boards) an int stores a 16-bit (2–byte) value. This yields a range of -32,768 to +32,767 (minimum value of -2^15 and a maximum value of (2^15)-1).
On the Arduino Zero, an int stores a 32-bit (4-byte) value. This yields a range of -2,147,483,648 to 2,147,483,647 (minimum value of -2^31 and a maximum value of (2^31)-1).
int stores negative numbers with a technique called 2's complement math. The highest bit, sometimes referred to as the "sign" bit, flags the number as a negative number. The rest of the bits are inverted and 1 is added.

unsigned int

unsigned int (unsigned integers) on the Arduino Uno and other ATMEGA based boards is the same as int in that they store a **2–byte** value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1).
The Arduino Zero stores a 4 byte (32-bit) value, ranging from 0 to 4,294,967,295 (2^32-1).
The difference between unsigned ints and (signed) ints, lies in the way the highest bit, sometimes refered to as the "sign" bit, is interpreted. In the Arduino int type (which is signed), if the high bit is a "1", the number is interpreted as a negative number, and the other 15 bits are interpreted with 2's complement math.

word

Word stores a 16-bit unsigned number, from 0 to 65535 (2^16-1). It is same as an unsigned int.

```
word w = 10000;
```

long

Long variables are extended size variables for number storage, and store 32 bits (4 bytes), from -2,147,483,648 to 2,147,483,647.
If doing math with integers, at least one of the numbers must be followed by an L, forcing it to be a long.

```
long speedOfLight = 186000L
```

unsigned long

Unsigned long variables are extended size variables for number storage, and store 32 bits (4 bytes). Unlike standard longs unsigned longs won't store negative numbers, making their range from 0 to 4,294,967,295 (2^32 - 1).

```
unsigned long time;

void setup() {
 Serial.begin(9600);
 }
void loop() {
Serial.print("Time: ");
time = millis(); //prints time since program started
Serial.println(time); // wait a second so as not to send massive
                      //amounts of data
delay(1000);
}
```

Float

Float is a datatype for floating-point numbers, a number that has a decimal point. Floating-point numbers are often used to approximate analog and continuous values because they have greater resolution than integers. Floating-point numbers can be as large as 3.4028235E+38 and as low as -3.4028235E+38. They are stored as 32 bits (4 bytes) of information.

```
int x;
int y;
float z;
x = 1;
y = x/2;
// y now contains 0, ints can't hold fractions
Z = (float)x/2.0;
// z now contains .5 (you have to use 2.0,not 2)
```

double

Double precision floating point number. On the Uno and other ATMEGA based boards, this occupies 4 bytes. That is, the double implementation is exactly the **same as the float**, with no gain in precision.
On the Arduino Zero, doubles have 8-byte (64 bit) precision.

String

Text strings can be represented in two ways. You can use the string data type, or you can make a string out of an array of type char and null-terminate it.

```
char Str1[15];
char Str2[8]  = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8]  = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ]   = "arduino";
char Str5[8]  = "arduino";
char Str6[15] = "arduino";
```

Generally, **strings are terminated with a null character** (ASCII code 0). This allows functions (like Serial.print()) to tell where the end of a string is. Otherwise, they would continue reading subsequent bytes of memory that aren't actually part of the string.

Array

An *array* is a collection of variables that are accessed with an index number. Arrays in the C programming language, on which Arduino is based, can be complicated, but using simple arrays is relatively straightforward.

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

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

To assign a value to an array:

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

To retrieve a value from an array:

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

Operators

Arithmetic operators

= (assignment operator)
+ (addition)
- (subtraction)
* (multiplication)
/ (division)
% (modulo)

Bitwise operators

& (bitwise and)
| (bitwise or)
^ (bitwise xor)
~ (bitwise not)
<< (bitshift left)
>> (bitshift right)

Compound Operators

++ (increment)
-- (decrement)
+= (compound addition)
-= (compound subtraction)
*= (compound multiplication)
/= (compound division)
%= (compound modulo)
&= (compound bitwise and)
|= (compound bitwise or)

++ (increment) /
-- (decrement)

Increment or decrement a variable

x++; // increment x by one and returns the old value of x
++x; // increment x by one and returns the new value of x
x-- ; // decrement x by one and returns the old value of x
--x ; // decrement x by one and returns the new value of x

Example

```
x = 2;
y = ++x;   // x now contains 3, y contains 3
y = x--;   // x contains 2 again, y still
           // contains 3
```

+= , -= , *= , /= , %=

Perform a mathematical operation on a variable with another constant or variable. The += (et al) operators are just a convenient shorthand for the expanded syntax, listed below.

Syntax

```
x += y;   // equivalent to the expression x = x + y;
x -= y;   // equivalent to the expression x = x - y;
x *= y;   // equivalent to the expression x = x * y;
x /= y;   // equivalent to the expression x = x / y;
x %= y;   // equivalent to the expression x = x % y;
```

x: any variable type

y: any variable type or constant

Example

```
x  = 2;
x += 4;        // x now contains 6
x -= 3;        // x now contains 3
x *= 10;       // x now contains 30
x /= 2;        // x now contains 15
x %= 5;        // x now contains 0
```

If statement

if, which is used in conjunction with a comparison operator, tests whether a certain condition has been reached, such as an input being above a certain number. The format for an if test is:

```
if (someVariable > 50)
{
// do something here
}
```

Comparison operators

x == y (x is equal to y)
x != y (x is not equal to y)
x < y (x is less than y)
x > y (x is greater than y)
x <= y (x is less than or equal to y)
x >= y (x is greater than or equal to y)

Logic operators

&& (and)
|| (or)
! (not)

If/else

if/else allows greater control over the flow of code than the basic **if** statement, by allowing multiple tests to be grouped together. For example, an analog input could be tested and one action taken if the input was less than 500, and another action taken if the input was 500 or greater. The code would look like this:

```
if (pinFiveInput < 500)
  {
    // action A
  }
else
  {
    // action B
  }
```

```
if (pinFiveInput < 500)
{
   // do Thing A
}
else if (pinFiveInput >= 1000)
{
   // do Thing B
}
else
{
   // do Thing C
}
```

switch … case statement

Like if statements, *switch...case* controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions.
In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run. The *break* keyword exits the switch statement, and is typically used at the end of each case.

Syntax of switch statement

```
switch (var)
{
  case label:
    // statements
    break;
  case label:
    // statements
    break;
  default:
    // statements
  break;
}
```

var: the variable whose value to compare to the various cases
label: a value to compare the variable to

Example

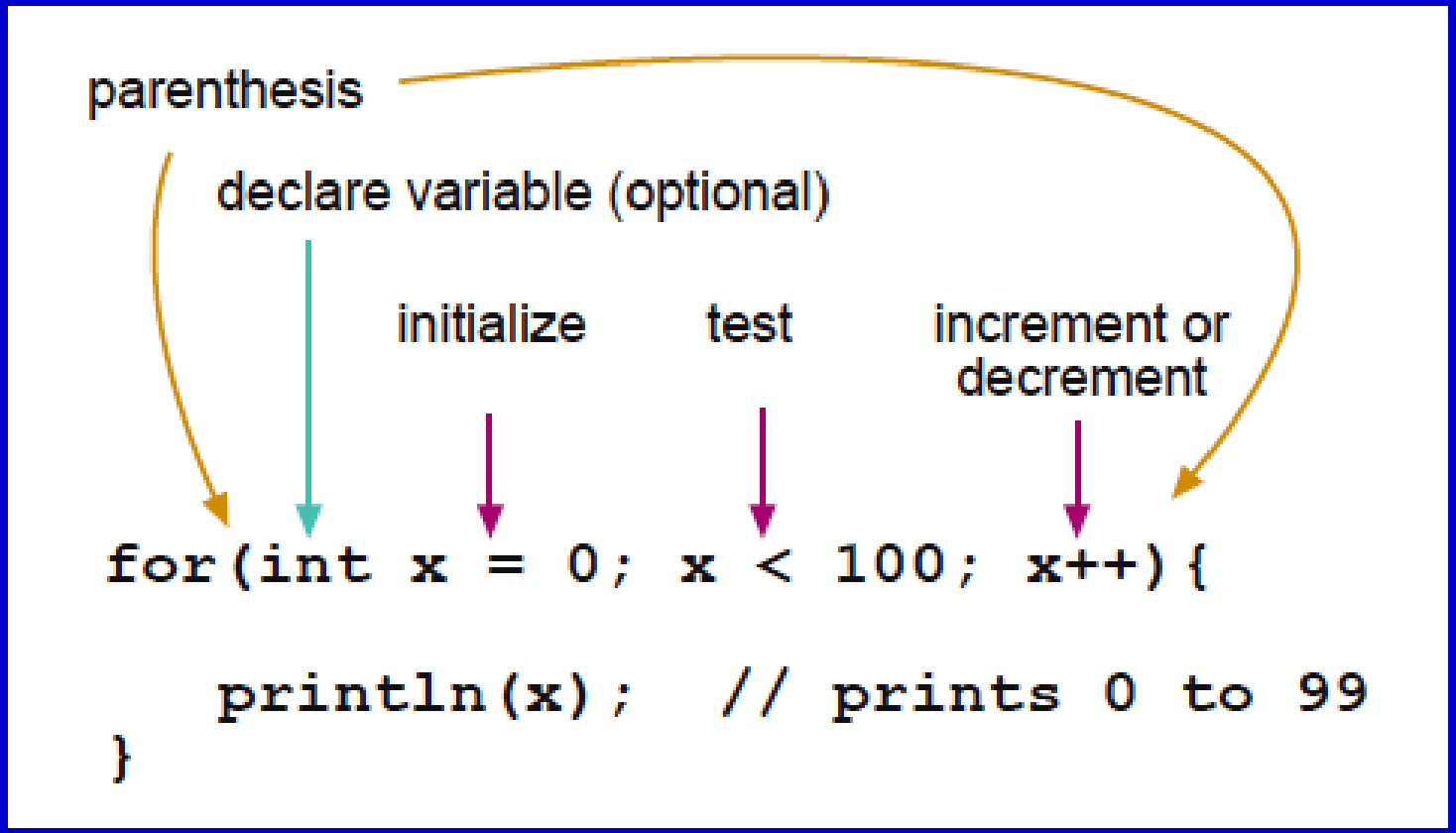
```
switch (var)
 {
    case 1:
      {
      //do something when var equals 1
      int a = 0;
      .......
      .......
      }
      break;
    default:
      // if nothing else matches, do the default
      // default is optional
    break;
  }
```

for

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. The *for* statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

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

Example of for statement



while

while loops will loop continuously and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the *while* loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

```
while(expression)
{
  // statement(s)
}
```

Example of while statement

```
var = 0;
while(var < 200)
 {
  // do something repetitive 200 times
  var++;
 }
```

do while

The *do* loop works in the same manner as the *while* loop, with the exception that the condition **is tested at the end** of the loop, so the *do* loop will *always* run at least once.

```
do
{
 // statement block
}
while (test condition);
```

```
 do
  {
   delay(50); // wait for sensors to stabilize
   x=readSensors(); // check the sensors
  } while (x < 100)
```

break

break is used to exit from a do, *for*, or *while* loop, bypassing the normal loop condition. It is also used to exit from a switch statement.

```
for (x = 0; x < 255; x ++)
{
 analogWrite(PWMpin, x);
 sens = analogRead(sensorPin);
  if (sens > threshold)
   { // bail out on sensor detect
   x = 0;
   break;
   }
   delay(50);
}
```

continue

The *continue* statement skips the rest of the current iteration of a loop (do, for, or while). It continues by checking the conditional expression of the loop, and proceeding with any subsequent iteration.

```
for (x = 0; x < 255; x ++){

     if (x > 40 && x < 120)
     {  // create jump in values
      continue;
     }
     analogWrite(PWMpin, x);
     delay(50);
}
```

return

return terminates a function and returns a value to the calling function, if desired.

```
int checkSensor()
{
    if (analogRead(0) > 400)
      {
       return 1;
      }
    else
      {
       return 0;
      }
}
```

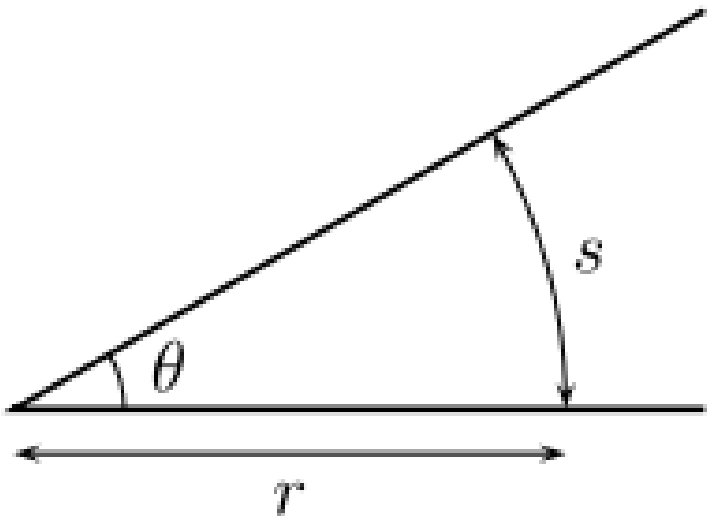
goto

goto transfers program flow to a labeled point in the program

Syntax

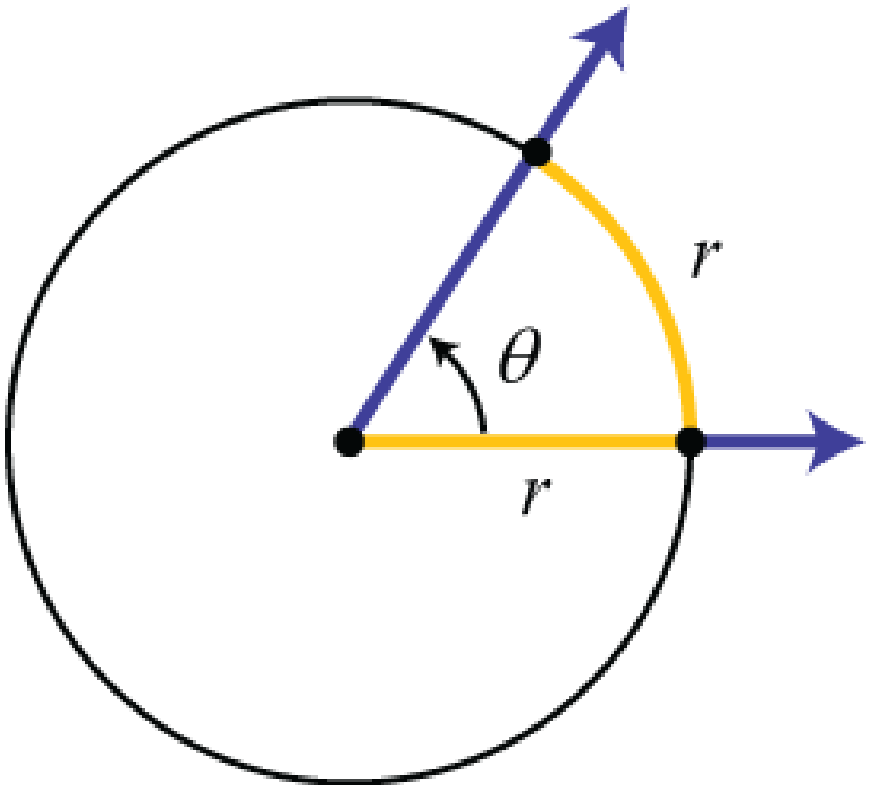
```
goto label; // sends program flow to the label
```

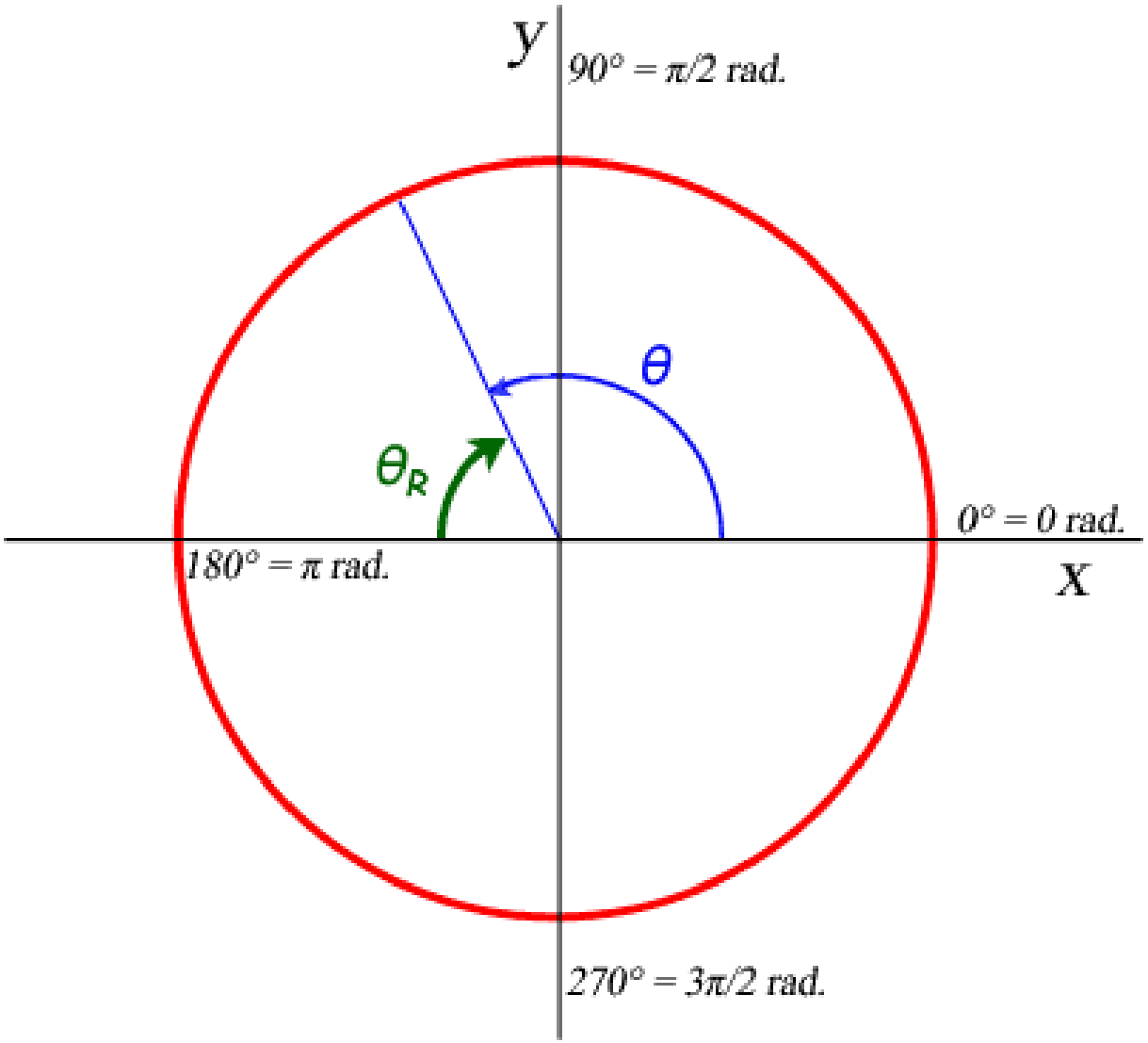
rad

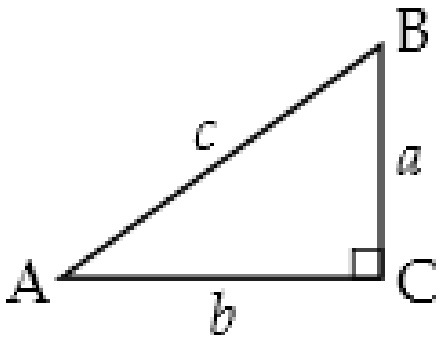


$$\theta = \frac{s}{r}$$

$\theta = 1$ radian (rad)




y
90° = π/2 rad.
θ
θR
0° = 0 rad.
180° = π rad.
X
270° = 3π/2 rad.



Pythagorean Theorem:

$$a^2 + b^2 = c^2$$

$$\sin A = \frac{a}{c} = \left(\frac{\text{opposite}}{\text{hypotenuse}} \right)$$

$$\cos A = \frac{b}{c} = \left(\frac{\text{adjacent}}{\text{hypotenuse}} \right)$$

$$\tan A = \frac{a}{b} = \left(\frac{\text{opposite}}{\text{adjacent}} \right)$$

sin(rad)

Η συνάρτηση *sin*(*rad*) υπολογίζει το ημίτονο (sine) μιας γωνίας σε rad. Το αποτέλεσμα θα είναι μεταξύ -1 και 1.

rad: the angle in radians (*float*)

Returns

the sine of the angle (*double*)

cos(rad)

Η συνάρτηση *cos*(*rad*) υπολογίζει το συνημίτονο (cos) μιας γωνίας σε rad. Το αποτέλεσμα θα είναι μεταξύ -1 και 1.

rad: the angle in radians (*float*)

Returns

the cosius of the angle (*double*)

char()

Converts a value to the char data type.

char(x)

x: a value of any type

byte()

Converts a value to the **byte data** type.

byte(x)

x: a value of any type

long()

Converts a value to the **long data** type.

long(x)

x: a value of any type

3.1. Να μετατραπούν (αναλυτικά) οι αριθμοί +85, -85 του δεκαδικού συστήματος αρίθμησης στους αντίστοιχους αριθμούς του δυαδικού συστήματος αρίθμησης σε παράσταση προσημασμένου συμπληρώματος του 2. Η παράσταση να γίνει με 8 bit.

3.2. Να βρεθεί αναλυτικά ποιους προσημασμένους αριθμούς του δεκαδικού συστήματος αρίθμησης παριστάνουν οι αριθμοί του δυαδικού συστήματος 10101011, 01100110 σε παράσταση προσημασμένου συμπληρώματος του 2.

3.3. Να βρεθεί ποιος είναι ο μέγιστος και ποιος ο ελάχιστος μη προσημασμένος αριθμός που μπορεί να παρασταθεί με 8 bit. Να βρεθεί επίσης ποιος είναι ο μέγιστος και ποιος ο ελάχιστος προσημασμένος αριθμός που μπορεί να παρασταθεί με 8 bit σε σύστημα συμπληρώματος του 2. Γενικεύσατε για n bit.

3.4. Να μετατραπεί ο αριθμός 01101110 του δυαδικού συστήματος στο δεκαεξαδικό σύστημα.

3.5. Να μετατραπεί ο αριθμός 3AF του δεκαεξαδικού συστήματος στο δυαδικό σύστημα.

3.6. Να μετατραπεί ο αριθμός 01101110 του δυαδικού συστήματος στο οκταδικό σύστημα.

3.7. Να μετατραπεί ο αριθμός 367 του οκταδικού συστήματος στο δυαδικό σύστημα.

3.8. Να μετατραπεί ο αριθμός 01101110 του δυαδικού συστήματος στο δεκαεξαδικό σύστημα.

3.9. Να μετατραπεί ο αριθμός 3AF του δεκαεξαδικού συστήματος στο δυαδικό σύστημα.

3.10. Να μετατραπεί ο αριθμός 01101110 του δυαδικού συστήματος στο οκταδικό σύστημα.

3.11. Να μετατραπεί ο αριθμός 367 του οκταδικού συστήματος στο δυαδικό σύστημα.

3.12. Να κωδικοποιηθούν οι χαρακτήρες J και j στον κώδικα ASCII.

3.13. Για τον κώδικα ASCII ποιο bit πρέπει να αλλάξουμε για να κάνουμε τα μικρά γράμματα του Αγγλικού αλφαβήτου κεφαλαία.

3.14. Να βρεθεί ποιοι χαρακτήρες του κώδικα ASCII είναι οι επόμενοι

1101110, 0111001, 0111000

3.15. Δώστε την διάταξη των bits που απεικονίζει τον δεκαδικό αριθμό 235 α) στο δυαδικό σύστημα β) στον κώδικα BCD (Binary Coded Decimal) γ) στον κώδικα ASCII (7 bits).

3.16. Να βρεθεί πιο bit της κωδικοποίησης των αλφαβητικών χαρακτήρων στον κώδικα ASCII πρέπει να αλλάξουμε για να μετατραπεί ένας μικρός Αγγλικός χαρακτήρας σε κεφαλαιο και αντίστροφα.

3.17. Υπολογίστε το μέγεθος σε ακτίνια των γωνιών 360, 270,180, 90, 60, 30, 15 μοιρών.

3.18. Υπολογίστε το ημίτονο των γωνιών 0, 360, 270,180, 90, 60, 30, μοιρών.

3.19. Περιγράψτε την δομή των προγραμμάτων του Arduino