ESP BASIC

# Language Reference

For ESP Basic 3.0 aXX

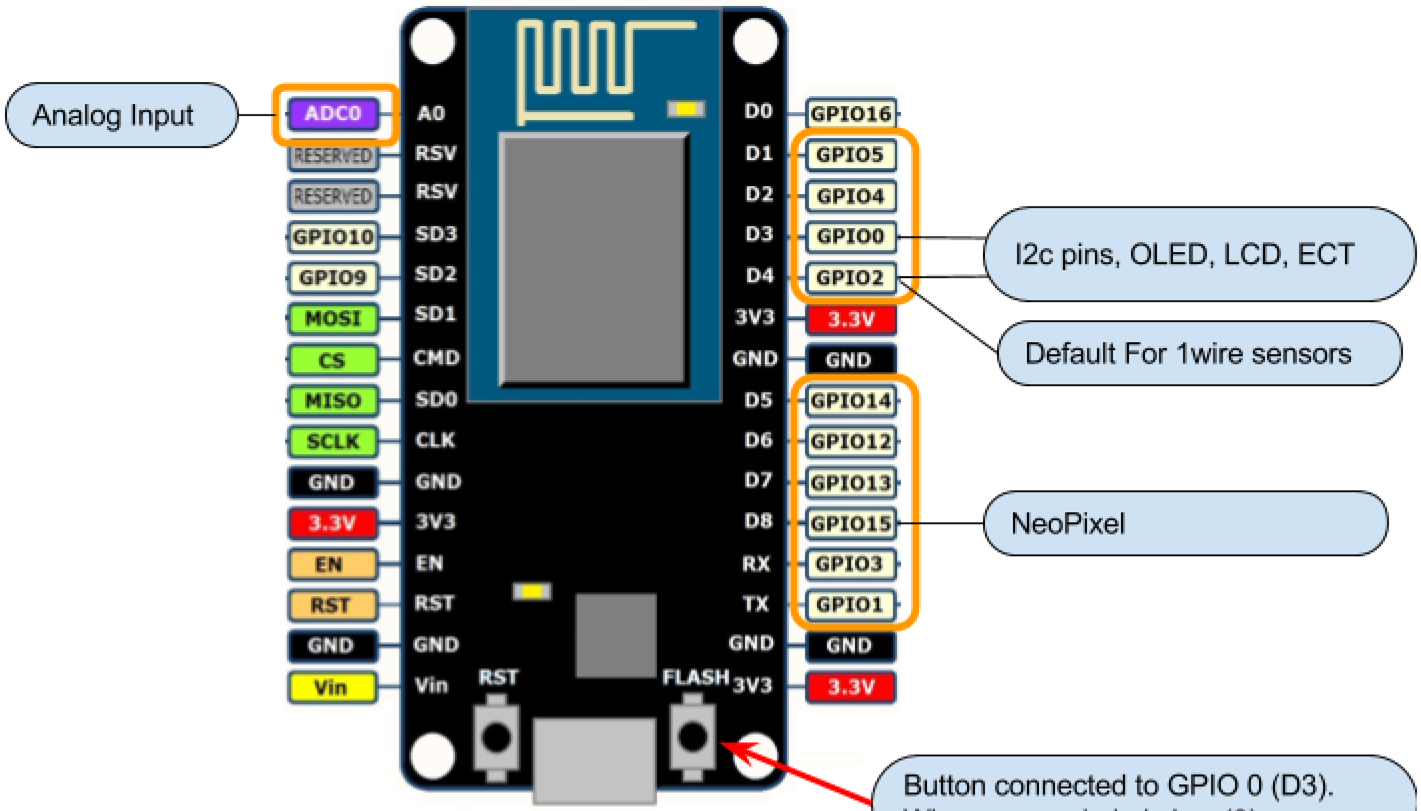
HTTP://ESP8266BASIC.COM

Onewire temperature sensor code contributed by Rotohammer.

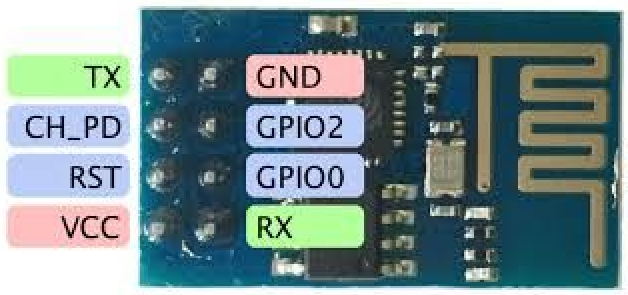
Significant speed improvements submitted by cicciocb
Uploading of bas files improvement added by cicciocb
JSON parsing routine added by cicciocb

# Table Of Contents

## Hardware Basics (Node MCU Board)



## ESP-01 BOARD



# BASE COMMANDS (Basic Core)

## IF .. THEN:

The if then command will compare 2 values or variables.
To check if strings are equal to each other use ‘==’, for numbers use ‘=’.
if {var or value} {=, ==,>,<,<>,>=,<=}, {var or value} then {statement to be executed if true} {else} {Statement if false}

If then statements can now be split between multiple lines. The format shown below can be used.

```
If bla = 5 then
   Print "Bla is 5"
Else
   Print "bla is not 5"
Endif
```

The if can be nested as below :

```
if a > b then
  print "a > b"
  if b >c then
    print "b > c"
  else
    print "b < c"
  endif
else
  print "a < b"
endif
```

## DO...LOOP:

Will allow you to establish a loop that will either run infinitely or only if specified conditions are true.

```
DO
...
LOOP {WHILE/UNTIL} {Condition to be evaluated}
```

If you want do the loop infinite:

```
Do
 Print "Looping for eternity"
Loop until 0
```

Do loop with until condition:

```
Do
 bla = bla + 1
 Print bla
loop until bla > 5
```

Do loop with while condition:

```
Do
 Bla = bla + 1
 Print bla
Loop while bla < 5
```

## LET:

Let will perform math or string operations on a set of values or variables and place the result into a variable.
To add strings together you must use the '&' symbol instead of the '+'
Can also be used to evaluate logic operations like in the if then statement. If the result is true it will return a -1 otherwise a 0

```
let {Result var} = {value or var} {operator *,/,+,-,^,&} {value or var}
let {Result var} = {value or var} {=, ==,>,<,<>,<=,>=} {value or var}
```

Can also be used without the let statement

```
bla = 5 + 5
```

| **Comparison Operators:** | **Math Operators:** |
|---|---|
| = Equal for numbers | + Add (Numbers Only) |
| == Equal for strings | - Subtract |
| > More than | * Multiply |
| < less than | / Divide |
| <> Not Equal | ^ To the power of |
| >= more than or equal | & Add (Text) |
| <= less than or equal | % Modulo |
| **Logic Operators:** | **Boolean Operators:** |
| And boolean 'and' (ex. If a=10 and b=20 then ..) | And boolean 'and' (ex. a = b and c) |
| Or boolean 'or' (ex. If a=10 or b=20 then ..) | Or boolean 'or' (ex. A = a or 16) |
| Xor boolean 'xor' (ex. If a=10 xor b=20 then ..) | Xor boolean 'xor' (ex. A = b xor 255) |
| Not boolean 'not' (ex. If not (a=10) then ..) | Not boolean 'not' (ex. A = not 15 ) |
| Any combination of these operators is allowed. | << shift left (ex. A = b << 4) |
| (ex. If (a=10 or b=20) and c=30 then | >> shift right (ex. A = b >> 8) |

## DIM:

Will dimension an array. Arrays dimensioned without the $ will be defined as numeric.
Arrays defined with a $ will be defined as string.
Arrays defines without the $ can be defined as string adding 'as string' at the end of the line.
Can be used just like other variables. Can not be used for gui widgets.

```
dim araynumeric( {variable or numeric value} )
dim arraystring$( {variable or numeric value} )
Dim arraystring( {variable or numeric value} ) as string
```

## UNDIM:

Will release memory used by an array

```
undim arraystring$
```

## GOTO:

Will jump to the branch label of your choice

```
goto {label}
```

## GOSUB AND RETURN:

Will jump to the branch label of your choice and allow you to return to the location you call the gosub from and continue executing code after the return command is executed. The gosub command has a stack of 255 return points so it is possible to gosub from inside a gosub.

```
gosub {label}
return
```

Example:

```
Print "Some text"
Gosub [mySub]
Print "Some more text"
end

[mySub]
Print "Text from in the gosub"
return
```

## FOR ... NEXT loops:

A for ... next loop will loop x number of times.

```
For x = 1 to 5
Print x
Next x
```

A for ... next loop with optional step value.

```
For x = 1 to 5 step 2
Print x
Next x
```

The step can be negative and not limited to integer.

```
For x = 5 to 0 step -0.1
Print x
Next
```

The variable after the next is optional.

## BRANCH LABELS:

A branch label allows you to jump from one part of the program to another. It can be a single string of characters with no spaces.
The  [ and ] are required to define a branch.
[label] (branch labels like in qbasic. No colon required)

## PRINT:

Will output text or a variable to the serial interface and the browser with a new line.
print {value or var}

## MEMCLEAR:

Will purge all of the variable used in a basic program from memory.
memclear

# SERIAL COMMANDS AND FUNCTIONS

## SERIALPRINT:

Will output text or a variable to the serial interface only. No new line or added.
serialprint {value or var}

## SERIALPRINTLN:

Will output text or a variable to the serial interface only. Will terminate with a new line.
serialprintln {value or var}

## BAUDRATE:

Will set the baudrate for serial communications.
baudrate {value or var}

## INPUT:

Will request input via serial from the user and place it into a variable. Variable doesn’t need to be declared prior to request.
input {optional string or var to display in prompt}, {value or var}

## SERIALINPUT:

Will get any data received from the serial port into the variable.
Useful in combination with the command serialbranch (see example below)
serialinput {variable}

## SERIALFLUSH:

Will clear the serial input buffer. Discards all characters stored in the buffer.
serialflush

## SERIALTIMEOUT:

Will cause the input command to return whatever is in the serial  buffer even if there is no CR LF. Specifying a value of 0 will disable the serial timeout. Any non-zero value will be the number of milliseconds it will wait before timing out on an input command.

```
serialtimeout {value or var}
```

## Serial.read.int()

Will return a single character from the serial buffer as an integer. Will grab next character each time is is used.

```
Bla = Serial.read.int()
```

## Serial.read.chr()

Will return a single character from the serial buffer as a string. Will grab next character each time is is used.

```
Bla = Serial.read.chr()
```

## serial.available()

Will return the number of characters in the buffer.

```
Bla = serial.available()
```

## SERIAL PORT 2:

A complete Serial Port #2 is available.
This is now a software port so we can freely define the pins for the TX and the RX.
The speed can be any value up to 115200. The format is fixed to 8N1.
The TX and RX can be any valid pin (except GPIO16).
If not defined, the default pins are GPIO2 (TX_pin) and GPIO12 (RX_pin)

### SERIAL2BEGIN:

Open the serial port 2 with the defined baudrate.
It is optionally possible to define the pins associated with the TX and RX
serial2begin {baudrate},{TX_pin},{RX_pin}
Example :
*serial2begin 115200, 5 , 4*

### SERIAL2END

Close the serial port 2.

### SERIAL2PRINT:

Will output text or a variable to the serial 2 interface only. No new line or added.
serial2print {value or var}

### SERIAL2PRINTLN:

Will output text or a variable to the serial 2 interface only. Will terminate with a new line.
serial2println {value or var}

### SERIAL2INPUT:

Will get any data received from the serial port 2 into the variable.
Useful in combination with the command serial2branch (see example below)
serial2input {variable}

## END:

Terminates the currently running program.

```
end
```

## DEBUGGER RELATED COMMANDS:

### DEBUGBREAK:

Will pause program execution if in debug mode. Can continue by hitting continue button.
Debugbreak

### debug.log():

Will send text to the debug log when in debug mode.
debug.log({string or var for txt})

# Timers And Interrupts

## TIMER:

The Timer command will allow you periodically run a branch. The branch must end with the wait command. Setting the timer to 0 will disable it.

To Set timer:
timer {value or var for wait time in milliseconds}, {branch label}

To Disable timer:
timer 0

The below example will check the status of the d3 pin (gpio 0) and will print a msg to the serial terminal each time it finds the pin high. If the button is pressed and the pin is pulled low it will disable the timer and display the msg that the timer has been disabled.

Example:

```
Print "timer example"
Timer 1000, [test]
Wait

[test]
If io(pi,d3) = 0 then
   Timer 0
   serialPrintln "timer has been disabled"
Else
   serialPrintln "Button connected to d3 (gpio 0) is high"
Endif
wait
```

## TIMERCB:

The Timercb command will allow you periodically run a sub branch and interrupt the currently running code. Once the timercb branch has run it will continue with what ever line of code it was last at allowing for a sub branch to be executed even if the program is in a loop. The branch must end with the return command. Setting the timer to 0 will disable it.

To Set timer:
timercb {value or var for wait time in milliseconds}, {branch label}

To Disable timer:
timercb 0

The below example will check the status of the d3 pin (gpio 0) and will print a msg to the serial terminal each time it finds the pin high. If the button is pressed and the pin is pulled low it will disable the timer and display the msg that the timer has been disabled.

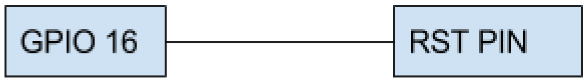
Example:

```
Print "timer example"
Timercb 1000, [test]
Wait

[test]
If io(pi,d3) = 0 then
   Timercb 0
   serialPrintln "timer has been disabled"
Else
   serialPrintln "Button connected to d3 (gpio 0) is high"
Endif
return
```

## SLEEP:

Will put device to sleep for a number of seconds. This will save power and allow for extended use on batteries.
Things to note. Sleep mode will cause a full reboot of the device. It will start the device and run the default program from the beginning. It takes 30 seconds before the device starts the default program after reboot.
GPIO16 needs to be tied to RST to wake from Sleep.



sleep {value or var in seconds}

To sleep for an hour look at the following example.
Example:

```
Sleep 3600   ‘60 seconds x 60 minutes
```

## REBOOT:

Will reboot the device upon execution.
reboot

## INTERRUPT:

Will execute a specific branch when a pins status changes. The interrupt polling only occurs when the esp is waiting and useful for things like push buttons trigger events. The branch must end with the wait command.
interrupt {pin no}, {branch label}
to disable an interrupt specify no branch label.
interrupt {pin no}

Example:

```
interrupt d3, [change]
wait

[change]
if io(laststat,d3) = 1 then print "Pin put high" else print "Pin put low"
wait
```

## SERIALBRANCH:

Defines a branch to be executed when serial input is received.
*serialbranch [branchLabel]*

*Example*

```
serialbranch [serialin]
wait

[serialin]
serialinput zz$
Serialprint "received:"
serialprintln zz$
return
```

## SERIAL2BRANCH:

Defines a branch to be executed when serial input is received on port #2.
*serial2branch [branchLabel]*

*Example*

```
serial2branch [serial2in]
wait

[serial2in]
serial2input zz$
Serial2print "received:"
serial2println zz$
return
```

# TELNET CLIENT

## TELNET.CLIENT.CONNECT():

Sets up a telnet client to the the indicated ip address and port. If no port is specified defaults to 23. Returns a 1 if successful and 0 if not.

```
Bla = telnet.client.connect({ip address of server}, {optional port no})
Ex.
Bla = telnet.client.connect("192.168.1.100", 1234)
Bla = telnet.client.connect("192.168.1.100") ‘defaults to port 23
```

## TELNET.CLIENT.AVAILABLE():

Returns the number of characters received in the buffer.

```
Bla = telnet.client.available()
```

## TELNET.CLIENT.READ.CHR():

Reads a single character from the buffer.

```
Bla = telnet.client.read.chr()
```

## TELNET.CLIENT.READ.STR():

Reads the entire buffer’s contents into a string.

```
Bla = telnet.client.read.str()
```

## TELNET.CLIENT.WRITE():

Sends a string to the server. Returns a 1 for success, 0 for failure

```
telnet.client.write({String to send})
Ex.
Bla = telnet.client.write("Hello world")
```

## TELNET.CLIENT.STOP():

Disconnects from the server.

```
telnet.client.stop()
```

## TELNETBRANCH:

Specifies a branch to launch when the esp receives characters from the server. Branch must end with the wait command.
Telnetbranch [branch]

# FILE I/O

## READ():

The read command allows you to retrieve information persistently stored in the flash memory of the ESP8266 module. This will return a string.

```
Bla = read({item name})
```

## READ.VAL():

The read command allows you to retrieve information persistently stored in the flash memory of the ESP8266 module. Will return a numeric value.

```
Bla = read.val({item name})
```

## WRITE():

The write command allows you to persistently store a data element in flash memory.

```
write({item name},{data to write})
```

## DEL.DAT():

Allows you to remove an element that has been written to flash memory. Use the same name that you used in the write command.

```
del.dat({item name})
```

## LOAD:

Will load another basic program into memory. Useful for chaining programs together.
Will load another program into memory and start executing it from the beginning.
All variables previously used will stay in memory and be available to the program that is loaded.
Useful for breaking a project up. ex.

```
LOAD "/myscript.bas"  or LOAD "/uploads/myscript.bas"
load {other program name}
```

# HARDWARE INTERFACE COMMANDS

For i2c functions see here.

## IO(PO,{PIN},{VALUE}):

po allows you to set pin on the esp high or low
io(po,{pin},{value})

## IO(PI,{PIN}):

Will place the high or low status of a pin into the variable chose.
Useful for reading push buttons and other electronic inputs.
P = io(pi,{pin})

## IO(PWO,{PIN},{VALUE}):

pwo allows you to set pin on the esp for PWM output
alternative as function
io(pwo,{pin},{value})

## PWMFREQ:

Allows the user to change the pwm frequency to a value  between 1 and 8000
Pwmfreq {FREQUENCY}

## IO(PWI,{PIN}):

Will place pwm input status of a pin in to the variable chose.
Useful for reading push buttons and other electronic inputs.
alternative as function
bla = io(pwi,{pin}

## io(SERVO,{PIN},{VALUE}):

Will set the angle of a servo connected to the the pin.
Angle must be between 0 and 180.
alternative as function

io(servo,{pin},{value})

## IO(AI):

Analog input is only available on the pin marked "ADC" on the ESP-12.
Useful for retrieving input from photoresistors and other devices.
As a function
Bla = io(ai)

## IO(LASTSTAT,{PIN}):

This will return the last polled status of a pin such as an interrupt and retain the value until it has changed again in another polling event. Also it will allow you to see the last value sent to a pin for servo, pwm or po.
io(laststat,{pin})

## TEMP({DEVICE ID}):

Will retrieve temperature sensor reading or sensor ROM codes from certain 1 wire devices connected to pin 2. With no arguments, all connected ROM codes are returned as a space separated string.  If argument is numeric, reading is returned from Nth connected device.  If argument is 16 character ROM code (as string) a reading is returned from that sensor.
Romcode = temp()
Bla = temp({Device ID})

## delay:

Will wait for a number of milliseconds before continuing execution.
Useful for making leds blink
delay {Var or value}

## GPIO1RESET():

Restore the normal behaviour of the serial port TX pin.
gpio1reset()
This is useful when the pin GPIO1 has been modified with commands like
po 1 0

## OLED & LCD DISPLAY COMMANDS

For i2c functions see here.

I2C pins are 0 and 2. On NodeMCU boards they are labeled as D3 and D4.

### OLEDPRINT:

Will print to the OLED display at specified position. X position for edge of display will differ for some models.
oledprint {String or var}, {x position}, {y position}

### OLEDCLS:

Will clear the screen.
oledcls

### OLEDSEND:

Will send a value to the display. This is for sending native commands and requires a byte as a number. Refer to OLED display command documentation from manufactures spec.
oledsend {Value or var}
1602 LCD COMMANDS

# OLED FUNCTIONS

These functions will only work on modules with 2 megs of flash or more.

## oled.color({1 or 0 for white or black})

Will set the color for the next item created on the screen.

## oled.cls()

Clears the screen on the oled display

## oled.line(x1,y1,x2,y2)

trace a line from the point (x1,y1) to the point(x2,y2)
example:
```
oled.line(10,10,20,20)
```

## oled.rect(x,y,w,h)

trace a rectangle from the point (x,y) with width 'w' and height 'h'
example:
```
oled.rect(10,10,20,20)
```

## oled.rect.fill(x,y,w,h)

trace a filled rectangle from the point (x,y) with width 'w' and height 'h'
example:
```
oled.rect.fill(10,10,20,20)
```

## oled.circle(x,y,rad)

trace a circle with center in (x,y), radius 'rad'
example:
```
oled.circle(50,120,20)
```

## oled.circle.fill(x,y,rad)

trace a filled circle with center in (x,y), radius 'rad’
example:
```
oled.circle.fill(50,50,20)
```

## oled.font({size})

Sets the font size to 10, 16 or 24.
Example:
```
oled.font(24)
```

## oled.print({string},{optional x},{optional y})

Print text on the display at the specified location.
Example:
```
oled.print("hello world", 1,1)
```

## LCD COMMANDS:

These are for i2c lcd displays.

### LCDPRINT:

Will print to the OLED display at specified position. X position for edge of display will differ for some models.
lcdprint {String or var}, {x position}, {y position}

### LCDCLS:

Will clear the screen.
lcdcls

### LCDBL:

Will turn the backlight on or off. 1 for on, 0 for off.
lcdbl {value or var}

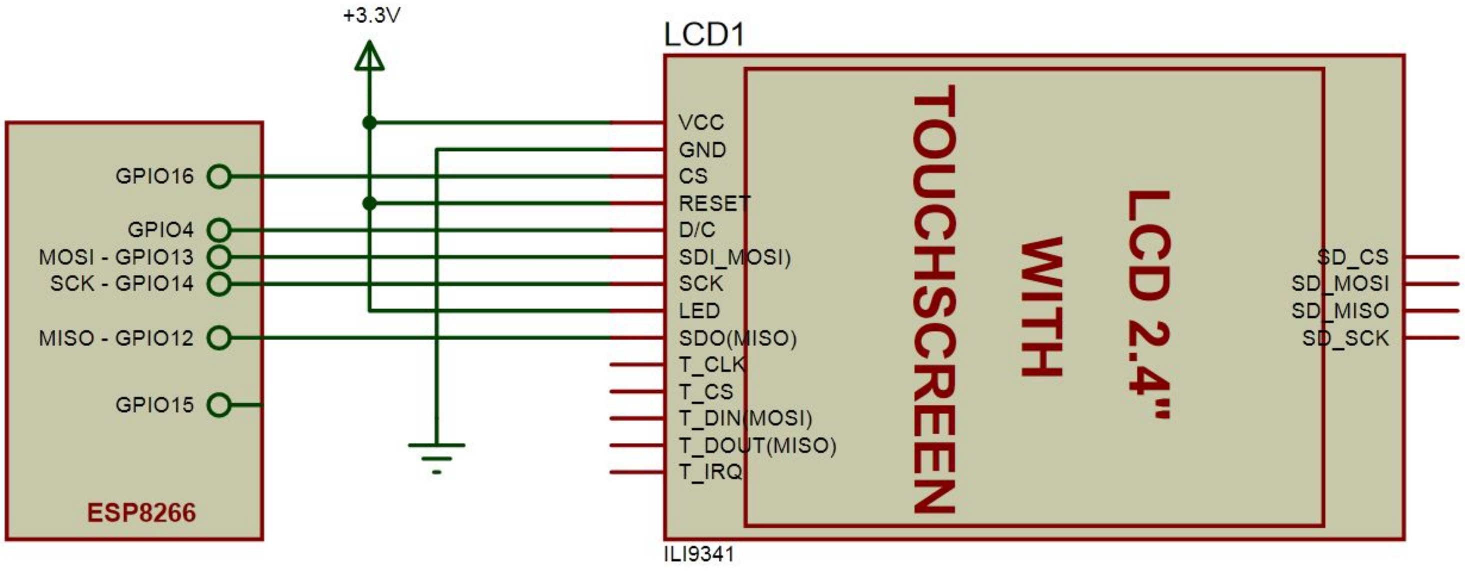
### LCDSEND:

Will send a value to the display. This is for sending native commands and requires a byte as a number. Refer to LCD display command documentation from manufactures spec.
MODE: 0=COMMAND, 1=DATA, 2=FOUR_BITS
lcdsend {Value or var to send}, {Value or Var for mode}

## TFT ILI9341 display

A TFT IL9341 SPI display can be connected to the ESP.
This display has a resolution of 320x240 with 16bits color.

The display requires at least 5 pins :
3 for the SPI, one for the CS and another one for the D/C



### tft.setup(CS_PIN, DC_PIN, rotation)

Initialise the display specifying the pin used for the CS, the DC and the screen rotation. If not specified, the rotation will default to 0 (portrait). Rotation can be 0 to 3
example:
tft.setup(16, 4, 3)      ‘set the pin 16 for the CS, the 4 for the DC and Landscape (3)

### tft.rgb(red, green, blue)

return the 16bits conversion of the colors red, green and blue
red, green and blue are in the range 0 - 255
example:
let col = tft.rgb(255,0,0)

## tft.fill(color)

Fill the screen with the color.
Example:

```
tft.fill(0)                  -> fill the screen with black
tft.fill(color)              -> fill the screen with the color;
                                color can be obtained from tft.rgb() function
tft.fill(tft.rgb(0,255,0))   -> fill the screen with green
```

## tft.cls()

Clears the screen on the TFT display

## tft.line(x1,y1,x2,y2,col)

trace a line from the point (x1,y1) to the point(x2,y2) with color 'col'
example:

```
tft.line(10,10,20,20,tft.rgb(255,0,0))
```

## tft.rect(x,y,w,h,col)

trace a rectangle from the point (x,y) with width 'w' and height 'h' with color 'col'
example:

```
tft.rect(10,10,20,20,tft.rgb(0,0,255))
```

## tft.rect.fill(x,y,w,h,col)

trace a filled rectangle from the point (x,y) with width 'w' and height 'h' with color 'col'
example:

```
tft.rect.fill(10,10,20,20,tft.rgb(0,0,255))
```

## tft.rect.round(x,y,w,h,rad,col)

trace a rounded rectangle from the point (x,y) with width 'w' and height 'h', radius 'rad' and with color 'col'
example:

```
tft.rect.round(10,10,100,100,5,tft.rgb(0,0,255))
```

## tft.rect.round.fill(x,y,w,h,rad,col)

trace a rounded filled rectangle from the point (x,y) with width 'w' and height 'h', radius 'rad' and with color 'col'
example:
```
tft.rect.round.fill(10,10,100,100,5,tft.rgb(0,0,255))
```

## tft.circle(x,y,rad,col)

trace a circle with center in (x,y), radius 'rad' and color 'col'
example:
```
tft.circle(160,120,50,65535)
```

## tft.circle.fill(x,y,rad,col)

trace a filled circle with center in (x,y), radius 'rad' and color 'col'
example:
```
tft.circle.fill(160,120,50,65535)
```

## tft.text.color(col)

define the color used for the text printed

## tft.text.cursor(x,y)

define the position of the cursor where the text will be printed

## tft.text.size(size)

define the size of the text printed; can be from 1 to 8 (the font is always the same)

## tft.print(txt)

print the text on the screen using the color, position and size defined with the previous commands

## tft.println(txt)

same as tft.print() but after the command goes to the next line

example:
```
tft.text.color(tft.rgb(255,255,0))
tft.text.cursor(0,50)
tft.text.size(2)
tft.print("This is ")
tft.println("print ")
tft.println("second line")
tft.println("third line")
```

## tft.demo()

runs a demo taken directly from the arduino demos. Useful at the beginning to check the display.

## tft.text.font(fontnumber)

Select the font to be used for the command print on TFT
The fontnumber can be :
0 : default fixed size 5x7 font
1 : Free Serif Bold 9  points
2 : Free Serif Bold 12 points
3 : Free Serif Bold 18 points
4 : Free Serif Bold 24 points
Example:
```
tft.text.font(2)     -> select Free Serif Bold 12 points
```

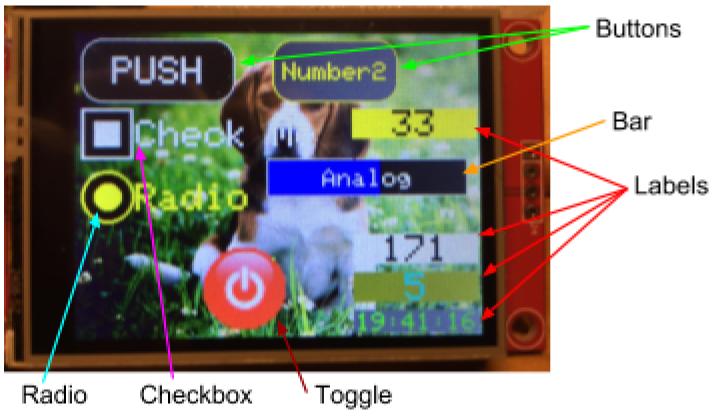
## tft.bmp(filename, x, y, backcolor)

Display a bitmap on the TFT screen.
Before use it, the file must be uploaded using the [FILE MANAGER] menu
Only non compressed bitmaps (.bmp) can be used (no gif, no png, no jpg).
The size of the image must be less than or equal to the size of the screen (max allowed size is 320 x 240).
If x and y are not defined, the position of the image will start at 0,0.
The back color will be used in case of background transparent images.
Example :
```
tft.bmp("/uploads/dog.bmp")
tft.bmp("/uploads/01d.bmp", 50,50, 0)
```

## TFT GUI OBJECTS

Example of Page created on the TFT using GUI commands

It is possible to implement some GUI objects on the TFT.
These objects are managed directly by ESP Basic and can be defined and modified with just some lines of code. They can also be linked to the touch screen events.



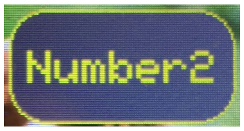
All the objects share a common set of properties :
**X, Y** : position of the object on the screen
**Width, Height** : dimensions of the object
**Forecolor, Backcolor:** colors of the object. If not defined by default are :
**Forecolor** = tft.rgb(255,255,0) - **Yellow** -
**Backcolor** = tft.rgb(100,100,100) - **Grey** -
**Label** : text associated to the object
**Textsize:** size of the text (actually is a multiplier of the standard font)
1 means normal size, 2 double size, and so on …
**Checked** : property associated to toggles / buttons / radio / checkbox.
Can be true or false
**Value:** property associated with the bar object.
It's a numeric value that can go from 0 to 100
**Icon1, icon2** : properties associated to the toggle object.
Define the filenames of the images associated to the object.
**Icon1** define the image when the **Checked** property is false
**Icon2** define the image when the **Checked** property is true
**Scale:** property associated to the toggle object.
Define the scaling (zoom) of the icons.
1 means normal size, 2 double size, and so on …

The objects are simply created with functions such as :
```
but1 = tft.obj.button("PUSH", 5,5,120,50,3,65535,0)
```
All the functions returns the ident of the object created (but1 in this example).
That ident will be useful when changing properties of the object or when identifying it within the touchscreen event.

## **tft.obj.button**(label, x, y, width, height, text_size, fore_color, back_color)

Will draw a button on the TFT. The function returns the ident of the object.
Example:

```
but2 = tft.obj.button("Number2", 155,5,100,50)
```



## **tft.obj.label**(label, x, y, width, height, text_size, fore_color, back_color)

Will draw a button on the TFT. The function returns the ident of the object.
Example:

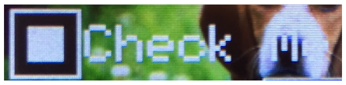
```
lab2 = tft.obj.label("19:41:16",218,220,100,18, 2, tft.rgb(0,255,0))
```

## **tft.obj.checkbox**(label, x, y, height, checked, text_size, fore_color, back_color)

Will draw a checkbox on the TFT. The function returns the ident of the object.
Example:

```
chk1 = tft.obj.checkbox("Check Me", 5,60,40,1, 3, 65535, 0)
```

The property checked (1 in this example) define the state of the checkbox.



## **tft.obj.radio**(label, x, y, height, checked, text_size, fore_color, back_color)

Will draw a checkbox on the TFT. The function returns the ident of the object.
example:

```
rad1 = tft.obj.radio("Radio", 5,110,40,1, 3)
```

The property checked (1 in this example) define the state of the radio.

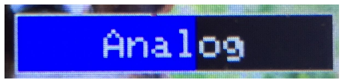
## tft.obj.toggle(icon1, icon2, x, y, checked, scale, back_color)

Will draw a toggle on the TFT. The function returns the ident of the object.
The toggle is an object that contains 2 icons, one associated to the false status and another associated to the true status.
Example:
*tog2 = tft.obj.toggle("/uploads/powgreen.bmp", "/uploads/powred.bmp", 100,170, 1, 2, -1)*

The property checked (1 in this example) define the state of the toggle.
The back_color define the back of the icons.
A color of -1 means transparent. Default is black.

## tft.obj.bar(label, x, y, height, width, text_size, fore_color, back_color)

Will draw a bar on the TFT. The function returns the ident of the object.
The bar (progress bar) it's an object that display a colored bar into a rectangular frame which size if function of the 'Value' property. Value can go from 0 to 100.
example:
*bar1 = tft.obj.bar("Analog",150,100,160,30, 2, 65535, tft.rgb(0,25,255))*
When created the bar value is always 0.



## Change the properties of the TFT GUI objects:

After the creation of objects, their properties can be changed with the following functions :

### **tft.obj.setlabel**(obj_ident, label)

Change the label of any object. The obj_ident is the value returned by the object creation functions.
Example:
*tft.obj.setlabel(lab2, mid(time(),12,8))*

### **tft.obj.setvalue**(obj_ident, value)

Change the value of the bar objects. The obj_ident is the value returned by the object creation functions. The value can be from 0 to 100.
Example:
*tft.obj.setvalue(bar1, io(ai)/10)*

### **tft.obj.setchecked**(obj_ident, check)

Change the checked property of the objects. The obj_ident is the value returned by the object creation functions.
Example:
*tft.obj.setchecked(but2, 1)      -> set the button to pushed state*
*tft.obj.setchecked(tog2, 0)      -> set the toggle to false state*

### **tft.obj.invert**(obj_ident)

Invert the checked property of the objects. The obj_ident is the value returned by the object creation functions.
Example:
*tft.obj.invert(but2)       -> invert the button state*
*tft.obj.invert(tog2)       -> invert the toggle state*

### **tft.obj.getlabel**(obj_ident)

Returns the label property of any object. The obj_ident is the value returned by the object creation functions.
Example:
*l = tft.obj.getlabel(lab2)*

## tft.obj.getvalue(obj_ident, value)

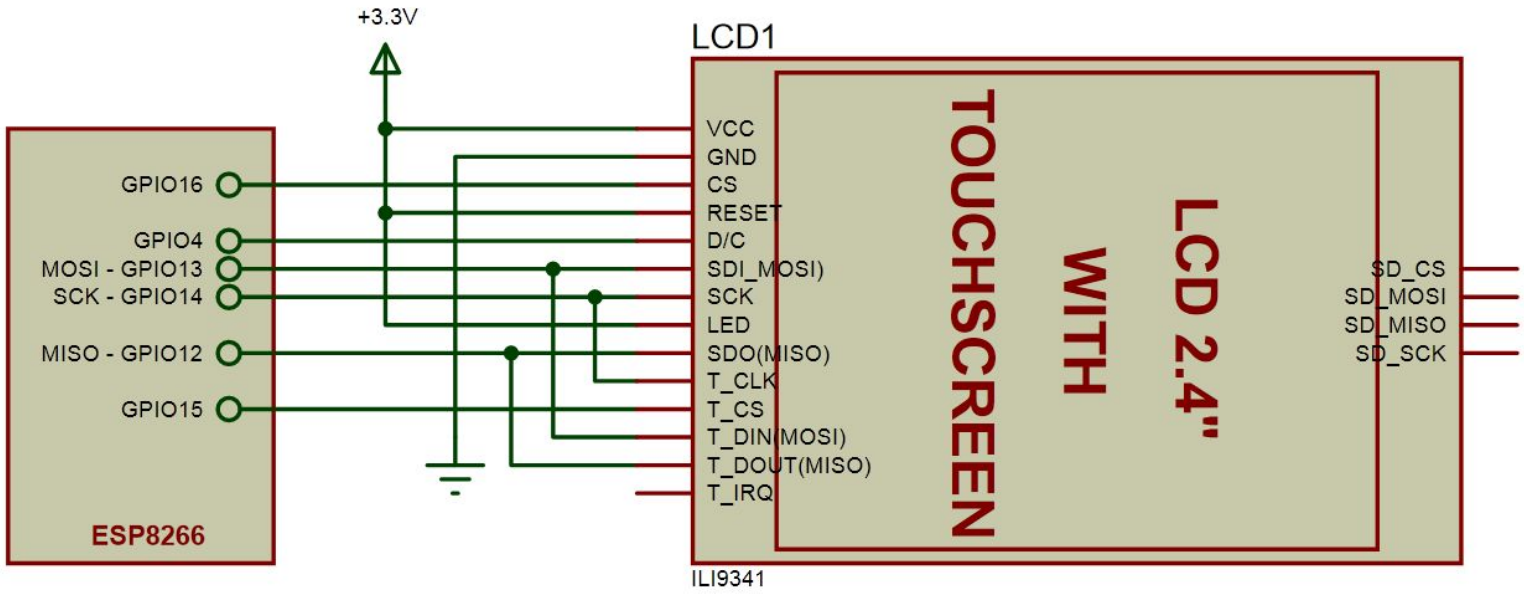
Returns the value of the bar objects. The obj_ident is the value returned by the object creation functions. The value can be from 0 to 100.
Example:
*v = tft.obj.getvalue(bar1)*

## tft.obj.getchecked(obj_ident, check)

Returns the checked property of the objects. The obj_ident is the value returned by the object creation functions.
Example:
*C = tft.obj.setchecked(but2)*

## TFT TOUCH SCREEN

It is possible to enable the touchscreen controller included into the ILI9341 TT. This controller is based on the chip XPT2046 and use SPI to communicate with the ESP module. The touch screen controller requires just one pin (for the T_CS signal). This pin can be freely defined using the tft.touch.setup(pin) command.



### **tft.touch.setup**(T_CS pin)

Enable the touch screen functionality. The T_CS pin needs to be specified as argument.
Example :
*tft.touch.setup(15)*

### **tft.touchx()**

Returns the X position of the touched position on the TFT screen.
Example:
*X = tft.touchx()*

### **tft.touchy()**

Returns the Y position of the touched position on the TFT screen.
Example:
*Y = tft.touchy()*

## tft.checktouch()

Returns the ident of the GUI object that has been touched. If the touched position is not within a defined object, the result will be -1.
This function is very useful in combination with the command **touchbranch**
Example:
*t = tft.checktouch()*

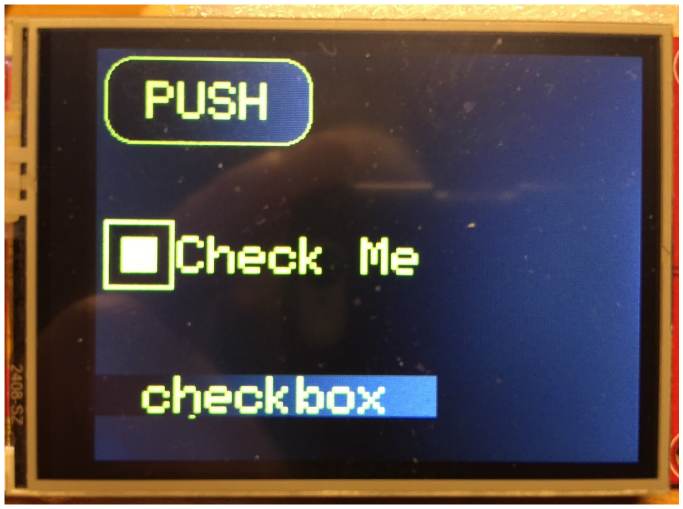
## touchbranch [label]

Defines a branch to be executed when the screen is touched.
The called function must terminate with then ’return’ command.
Example:
*Touchbranch [touchme]*

## Complete example:

```
tft.setup(16, 4, 3)
tft.touch.setup(15)
but1 = tft.obj.button("PUSH", 5,5,120,50,3)
chk1 = tft.obj.checkbox("Check Me", 5,100,40,1, 3)
lab1 = tft.obj.label("press", 0,190,200,24,3)
touchbranch [touchme]
wait

[touchme]
touch_obj = tft.checktouch()
serialprintln "checktouch " & touch_obj
if touch_obj = but1 then
  tft.obj.invert(but1)
  tft.obj.setlabel(lab1, "button")
endif
if touch_obj = chk1 then
  tft.obj.setlabel(lab1, "checkbox")
  tft.obj.invert(chk1)
endif
return
```



## FUNCTIONS (NEOPIXEL WS2812)

To use NEO Pixel strips you must connect the signal to pin 15, D8 for node mcu boards.

### NEO.SETUP():

Will configure the output pin to one other than the default.
neo.setup({pin NO})

### NEO():

Will set the desired pixel led to the color specified with an RGB (Red, Green, Blue) value. You can optionally disable instant writing to the neo pixel by using a 1 in the last parameter. This will allow for many pixels to updated all at once. Otherwise function will write instantly.
neo({LED NO},{R},{G},{B},{optional.  Set to 1 disable auto write})

```
Example
neo(5,250,250,250)     ‘ write to pixel instantly

neo(5,250,250,250,1)   ‘Add to buffer to be sent to neopixels.
```

### NEO.CLS():

Will turn off all the leds on the strip.
neo.cls()

### NEO.STRIPCOLOR():

Will set a range of leds to the desired color.
neo.stripcolor({start pixel},{end pixel},{R},{G},B})

### NEO.SHIFT():

Will display a shift affect. Will move all the pixels from 0 to 20 up one. This allows some nice scrolling animations with very little effort.
neo.shift({start pixel},{end pixel},{direction, 1 for forward, -1 for reverse.})

## NEO.HEX():

```
neo.hex({pixel data],{start pixel],int{Position in pixel data},{number
pixels},{brightness 0-100})
```

Example:

```
neo.hex("400fff",0,0,2,10)
```

Will use the string , starting at the start of the string copy it to the start of the neopixel display, copy 2 pixels, at 10% brighness.

# FUNCTIONS (DHT)

Functions to interface with the dht21 temp/humidity module.

## DHT.SETUP({model}, {pin}):

Allows specification of the model (11, 21, 22) and the pin for the DHT sensor.

```
dht.setup({model}, {pin})
```

## DHT.TEMP():

Will return the temperature reading in celsius from the sensor.

```
Bla = dht.temp()
```

## DHT.HUM():

Will return the humidity

```
Bla = dht.hum()
```

## DHT.HEATINDEX():

Compute heat index in Fahrenheit.

```
Bla = dht.heatindex()
```

## WEB INTERFACE COMMANDS

### CLS:

Will clear the screen and GUI buffer
cls

### WPRINT:

Will print text to the browser. Text will be sent to the browser upon the wait command. Note that there is no new line after it is printed so you must add html to create a new line or horizontal rule.
wprint {value or var}

### WPRINT with htmlvar() function:

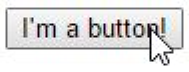
To place a dynamic variable that will update on each refresh of the page with the current contents of that variable use the htmlvar function.
Each time browser is refreshed the latest contents of the variable are place in the page.
wprint htmlvar({var name})

### IMAGE:

Will insert an image into the web page. Image file should be uploaded to device using the file manager. If you specify the image file name starting with http:// you can point it to an image stored on a 3d part web server saving space on the ESP file system.
image {image file name}

### BUTTON:

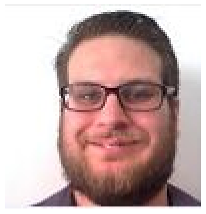
Functions like a goto command. Will be sent to the browser on the wait command.
Will goto the branch label when clicked in the browser
button {value or var}, {Branch Label}



## IMAGEBUTTON:

Functions like a goto command. Will be sent to the browser on the wait command. Will goto the branch label when clicked in the browser. The image file must be uploaded to the device using the file manager.

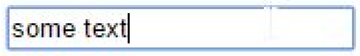
imagebutton {image file name}, {Branch Label}

(example of a spectacular looking image button)

## TEXTBOX:

The text inside the text entry filled will be whatever is in the variable. Will be displayed in the browser on the wait command

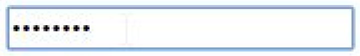
textbox {var name}



## PASSWORDBOX:

The text inside the password entry filled will be whatever is in the variable. Will be displayed in the browser on the wait command

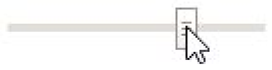
passwordbox {var name}



## SLIDER:

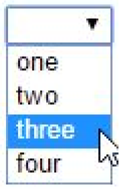
The slider will be created with a maximum and minimum value.

slider {Var name}, {min value}, {max value}



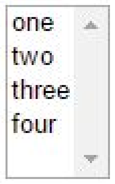
## DROPDOWN:

The selected item will be placed into the variable.
dropdown {var name}, {Item list separated by commas}
example:
dropdown bla, "One,Two,Three"



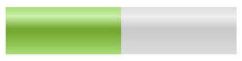
## LISTBOX:

The selected item will be placed into the variable.
listbox {var name}, {Item list separated by commas}, {Height in items}
example
listbox bla, "One,Two,Three", 5



## METER:

Will insert an html meter element into the page. Will update when the variable is changed.
Meter {var name}, {min value}, {max value}
Example:
x = 50
meter x, 0, 100



## ONLOAD:

Optional branch to be executed any time a page is loaded from the device. This code will be executed before the page is returned but after any branches for buttons. Branch must terminate with wait command.

```
onload {branch label}
```

## WAIT:

Sends all the accumulated gui commands to the browser.
The browser will display once this command is run. The esp will be placed in a wait state so that user interaction such as button press from the browser can be executed or pin interrupt functions. The program will halt unter there is user interaction.

```
Wait
```

## RETURNGUI:

Sends all the accumulated gui commands to the browser.
The browser will display once this command is run. The program will continue executing the next lines after this command. This command will refresh the entire contents of the browser window with any updates or changes made to variables using the htmlvar() function.

```
returngui
```

## HTMLID():

The html id function will return the randomly generated id for the last gui object created. Useful for javascript interaction capabilities.

```
htmlid()
```

## GUIOFF:

Guioff will purge the html buffer and prevent any additional data from being added with gui commands such as print, wprint, button ect. Also disables web sockets. Good for programs requiring very fast responses from interrupts, ect.

```
guioff
```

## GUION:

Turns the gui features back on if they were disabled with GUIOFF.

```
Guion
```

## CSSID:

The cssid command will apply css to the desired item by id. This should be executed immediately after creating a gui object.
Example:

```
cssid htmlid(), "background-color: yellow;"
```

## CSSCLASS:

The cssclass command will apply css to the desired class of gui objects.
Example: Will set all buttons to color yellow.

```
cssclass "button", "background-color: yellow;"
```

## JAVASCRIPT:

Will allow you to include javascript files in your page. File must be uploaded to the device using the file manager.

```
javascript {filename}
```

## CSS:

Will allow you to include css files in your page. File must be uploaded to the device using the file manager.
See the CSS example here.

```
css {filename}
```

## JSCALL:

Will allow for execution of javascript in the web browser for the purpose of calling a function is a javascript file included with the javascript command.

```
jscall {function name}
```

Example:

```
Javascript "beep.js"  'include the js file in the web page
Jscall "beep"         'call the beep function from the javascript loaded in the page.
```

## WEB MSG HANDLER COMMANDS

Msg handler events will only be handed when there is a request at the url
http://device ip/msg? is used.
See example /msg-url-usage.html for more information.

### MSGBRANCH:

Will set the branch for handling msg input urls. This branch will execute when the url "ESP-IP-address/msg?" is accessed.
msgbranch {branch label}

### MSGGET({url arg}):

Will retrieve a url argument and place it in a variable.
Bla = msgget({url arg})
Example:
Will retrieve the url argument "color" and place it into myColorVar
Use browser to access "ESP-IP-address/msg?color=blue"
myColorVar = msgget("collor")

### MSGRETURN:

Will set the text to be returned by the browser. No additional html is provide. Will overwrite previous return text if called more than once. Text will be returned when interpreter encounters the  wait command.
msgreturn {variable name or string for return}

## SMTP EMAIL COMMANDS

You will need an SMTP server such as http://www.smtp2go.com/
SMTP Server: mail.smtp2go.com
Port: 2525

For sending sms message to phones look up your carrier's sms gateway.
http://martinfitzpatrick.name/list-of-email-to-sms-gateways/

### SETUPEMAIL:

Will configure the email server. Requires the server, port, user name and password.
setupemail {server}, {port}, {user name}, {password}

### EMAIL:

Will send an email using the from and to address using the subject and body.
email {String To email}, {String From Email}, {String Subject}, {String Body}

# WEB GRAPHICS COMMANDS

Colors are optional. If no color is specified it will default to black
For examples you can look at /graphics-example.html and /graphic-clock-example.html

## Color palette

0 = Black
1 = Navy
2 = Green
3 = Teal
4 = Maroon
5 = Purple
6 = Olive
7 = Silver
8 = Gray
9 = Blue
10 = Lime
11 = Aqua
12 = Red
13 = Fuchsia
14 = Yellow
15 = White

## GRAPHICS

Adds a graphics element to the page
Each parameter can be a variable or a value.

```
graphics {width}, {height}
```

## GCLS:

Will clear the graphics buffer.

```
gcls
```

## LINE:

Creates a line in the graphic element
Each parameter can be a variable or a value.
line {x1}, {y1}, {x2}, {y2}, {color}

## CIRCLE:

Creates a circle in the graphic element
Each parameter can be a variable or a value.
circle {x1}, {y1}, {radius},  {color}

## ELLIPSE:

Creates a ellipse in the graphic element
Each parameter can be a variable or a value.
ellipse {x1}, {y1}, {radiusX}, {radiusY}, {color}

## RECT:

Creates a rectangle in the graphic element
Each parameter can be a variable or a value.
rect {x1}, {y1}, {radiusX}, {radiusY}, {color}

## TEXT:

Creates text in the graphic element
Each parameter can be a variable or a value.
text {x1}, {y1}, {Text to be displayed}, {angle}, {color}

# WIFI COMMANDS

## WIFI.CONNECT():

Will connect you to the wifi network using the ssid and password. Optionally you may specify the ip ADDRESS, Gateway and subnet mask for a static ip.
With just SSID
```
wifi.connect( {SSID} )
```
With SSID and PASSWORD:
```
wifi.connect( {SSID}, {password})
```
Static ip:
```
wifi.connect( {SSID}, {password}, {ip}, {gateway},  {netmask} )
```

## WIFI.AP():

Will create an access point with the name and password you specify.
If the password is not specified it will be an open network.
Open Network:
```
wifi.ap( {SSID} )
```
With SSID and PASSWORD:
```
wifi.ap( {SSID}, {password})
```

## WIFIOFF:

Will turn off all networking for sta and ap mode. The WiFioff command can be called in order to disable all networking. The ap or connect command if run after WiFioff will activate that particular WiFi mode. You can turn them both back on at any point with each of these commands.
```
Wifioff
```

## WIFIAPSTA:

Will allow for the connect and ap commands to be used simultaneously to connect to a network and broacast out an access point at the same time.
```
wifiapsta
```

## WIFI.SCAN():

Will return the number of available wifi networks. See an example here.
NumberOfNetworks = wifi.scan()

## WIFI.SSID():

Returns the network name for the selected network. See an example here.
Must run a wifi.scan() first.
wifi.ssid({Var for network number})

## WIFI.RSSI():

Returns the signal strength for the selected network. See an example here.
Must run a wifi.scan() first.
wifi.rssi({Var for network number})

## WIFI.BSSID():

Returns the mac address for the selected network. See an example here.
Must run a wifi.scan() first.
wifi.bssid({Var for network number})

## FUNCTIONS (THINGSPEAK API)

To get started with thing speak visit there web site at https://thingspeak.com/
You need to have an account and channel/api key all set up.
These functions will only work if the esp is connected to a network with the internet.

### SENDTS:

Will post the fields contents to the thingspeak service. Must use the thingspeak key and field number.

```
SENDTS({KEY},{FIELD NUMBER},{FIELD CONTENTS})
```

### READTS:

Will return the last value published for the desired field. Must use the thingspeak key, channel id and field number.

```
Bla = readts({KEY},{CHANNEL ID},{FIELD NUMBER})
```

# FUNCTIONS String

Functions can be called in place of variables and will typically return value.
Function names are not case sensitive. ie. LeN() and len() are both valid

## len():

Will return the length of the string
Bla = len({string or var name})

## instr():

Will return location of a sub string within a string.
A 3rd optional argument permit to sets the starting position for each search. If omitted, search begins at the first character position.
Bla = instr({original string},{string to locate},{optional start})

## instrrev():

Returns the position of the first occurrence of one string within another, starting from the right side of the string.
A 3rd optional argument permit to set starting position for each search, starting from the left side of the string. If Start is omitted then -1 is used, meaning the search begins at the last character position. Search then proceeds from right to left.
Bla = instrrev({string or var name},{string or var name to locate},{start})

## mid():

Will return the string from the start position to the end position inside of the input string.
Bla = mid({string or var name},{Start position},{number of characters})

## left():

Will return a string that contains a specified number of characters from the left side of a string.
Bla = left({string or var name},{length})

## right():

Will return a string containing a specified number of characters from the right side of a string.
Bla = right({string or var name},{length})

## trim():

Will return a string with leading a trailing spaces trimmed from it.
Bla = trim({string or var name})

## str():

Will return the returns a string representation of a number
Bla = str({number or var name})

## replace():

Will return a string after replacing the search text with the replacement text.
Bla = replace({string or var name},{string to search for},{replacement for string})

## chr():

Will return a character for the given number.
Bla = chr({string or var name})

## asc():

Will return a number for the first character in a string.
Bla = asc({string or var name})

## upper():

Will return the supplied string in UPPERCASE.
Bla = upper({string or var name})

## lower():

Will return the supplied string in lowercase.
Bla = lower({string or var name})

## id():

Will return the unique id of the chip.
Bla = id()

## version():

Will return the version string for the basic interpreter build.
Bla = version()

## word():

This function returns the nth word in the string, string variable or string expression.
The string delimiter is optional. When it is not used, the space character is the delimiter.
Bla = word( {original string}, {number}, {optional delimiter. Default is space} )
Ex.
*word("The quick brown fox jumped over the lazy dog", 5) will return "jumped"*

## json():

Will parse a json string for the articular named data element within it.
json({string or var name for data to be parsed},{string or var name for key name in data})
The key can have the following syntax :
"Key.subkey.innerkey….." . Array can also be included such as
"weather[5].description"

Example with OpenWeatherAPI :

```
let apid = "xxxxx" ‘ place your APP_ID here
let query = "api.openweathermap.org/data/2.5/weather?&units=metric&q=Miami,us&appid=" &
appid
let ret = wget(query)
serialprintln ret
let desc = json(ret,"weather.description")
let temp = json(ret,"main.temp")
let press = json(ret,"main.pressure")
let humid = json(ret,"main.humidity")
```

## ReadOpenWeather():

Will parse a json string coming from an OpenWeatherAPI forecast request.
At this kind of request generates a big amount of data, this function parse directly from the stream minimising the use of the RAM memory.
It permit to extract, from the stream, the particular item required; the extracted data can then be parsed by the function json().
The index_number can be 0, in that case the function will return the json root
readopenweather({String or var name for url} , {index_number})

Example with OpenWeatherAPI forecast :

```
let apid = "xxxxx" ' place your APP_ID here
'This query represent a daily forecast for Miami, Florida; each item represent a day
let query =
"api.openweathermap.org/data/2.5/forecast/daily?&units=metric&q=Miami,us&appid=" &
appid
let ret = readopenweather(query,1) ' the 1 means the first item of the list - 0 means
the root
let temp_min = json(ret,"temp.min")
let temp_max = json(ret,"temp.max")
let tim = json(ret,"dt")
let tim = unixtime(tim)
print "Date :" & tim & " T min " & temp_min & " T max " & temp_max
```

## FUNCTIONS (Numeric)

Functions can be called in place of variables and will typically return value.
Function names are not case sensitive. ie. LeN() and len() are both valid

Math Functions:

### sqr():

Will return the square root.
sqr({value or var name})

### sin():

Will return the sine of an angle.
sin({value or var name})

### cos():

Will return the cosine of an angle.
cos({value or var name})

### tan():

Will return the tangent of an angle.
tan({value or var name})

### log():

Will return the log of the provided value.
log({value or var name})

## rnd():

Will return a random number up to the value set.
rnd({value or var name})

## millis():

Will return number of milliseconds since boot time.
millis()

## int():

Will return an integer value.
int({string or var name})

## val():

Will return a value from a string.
val({string or var name})

## oct():

Will return the oct value of an integer.
oct({string or var name})

## hex():

Will return the hex value of an integer.
An optional 2nd argument define the number of digits
hex({string or var name},{nb_of_digits})
Example:

```
print hex(12)        -> c
print hex(12, 2)     -> 0c
print hex(2170)      -> 87a
print hex(2170, 4)   -> 087a
print hex(2170, 6)   -> 00087a
```

### hextoint():

Will return an integer value from an hex value
hextoint({string or var name})

### ramfree():

Will return the amount of ram free in bytes.
ramfree()

### flashfree():

Will return the amount of flash free in bytes. This is only applicable to flash designated for the file system.
flashfree()

## Function inherited from the parser (c++ style)

### pow(x, y ):

Returns x raised to the power of y.

### exp(x):

Returns the value of e raised to the xth power.

### asin(x):

Returns the arc sine of x in radians.

### acos(x):

Returns the arc cosine of x in radians.

### atan(x):

Returns the arc tangent of x in radians.

## atan2(x, y):

Returns the arc tangent in radians of y/x based on the signs of both values to determine the correct quadrant.

## abs(x):

Returns the absolute value of x converted to integer.

## fabs(x)

Returns the absolute value of x.

## floor(x):

Returns the largest integer value less than or equal to x.

## ceil(x):

Returns the smallest integer value greater than or equal to x.

## round(x):

Return the integral value nearest to x rounding halfway cases away from zero, regardless of the current rounding direction.

# FUNCTIONS (Time&Date)

Time and date functions:

## time():

Will return the date and time as a string. You can optionally specify the format you want it back in.
Options: month, day, hour, min, sec, year, dow (Day of the week, ex. fri)
time({Optional specification of format})
with an option.
time("year")
time("year-month-day") will return "2016-Apr-10"

## unixtime():

Permit to convert from a date/time in unix format (a number) to text.
Will return the date and time as a string. You can optionally specify the format you want it back in.
Options: month, date, hour, min, sec, year, dow (Day of the week, ex. fri)
unixtime({datenumber} , {Optional specification of format})
with an option.
unixtime(1459681200, "year"

## time.setup():

Will set up the time time zone and daylight savings attribute.
time.setup({number or var for time zone},{number or var for dst},{optional server})

# FUNCTIONS (Web Related)

Internet functions:

## wget():

Will fetch the html contents of a web page and return it as a string.
Do not put "http://" in front of the url. Defaults to port 80 if none is specified.
wget({String or var name for url},{Optional port number})

## ping():

Will return a 1 if endpoint is reachable and a 0 if not. Will only work in 4 meg version.
Bla = ping("192.168.1.1")
Bla = ping("google.com")

## ip():

Will return the units ip address as a string.
ip()

# FUNCTIONS I2C

## i2c functions:

For more information on usage look at this example.

## i2c.begin():

Will begin transmission to the I2C device with desired address.
i2c.begin({value or var for device address})

## i2c.setup():

Will change the default pins for the i2c interface.
i2c.setup({SDA}, {SCL})

## i2c.write():

Will write a single value (1 character) to the i2c device.
i2c.write({value or var for data})

## i2c.end():

Will terminate the i2c contamination with a particular device.
i2c.end()

## i2c.requestfrom():

Will request a quantity of bytes from  device.
i2c.requestfrom({value or var for device id},{value or var for number of bytes to request})

## i2c.available():

Returns the number of bytes available for retrieval with i2c.read().
i2c.available()

## i2c.read():

Will return a single character as an integer. Character returned will be next out of buffer.

```
i2c.read()
```

## FUNCTIONS SPI

The SPI port (H/W) is available for general use.
The following pins are defined :
SCK : GPIO14
MISO: GPIO12 (Master Input, Slave Output) (input  for the ESP8266)
MOSI: GPIO13 (Master Output,Slave Input ) (output for the ESP8266)

The CS pin can be any pin. (controlled by user program)

### spi.setup():

Opens the SPI port and set the speed.
It possible to optionally define the SPI_MODE and the data direction.
By default these values are SPI_MODE0 and MSB_FIRST. Only spi modes 0 and 1 are supported.

spi.setup({speed}, {MODE}, {MSB_FIRST})
Example :

```
spi.setup(1000000)        -> 1Mbit/sec, mode 0, MSB_FIRST
spi.setup(500000, 1, 1)   ->500Kb/sec, mode 1, MSB_FIRST
spi.setup(100000, 1, 0)   ->100Kb/sec, mode 1, LSB_FIRST
```

### spi.byte():

Write and receive byte at the same time (the arguments are numbers)
rec = spi.byte({send variable})
The value received and sent are numbers (byte)
Can be used as a command if the received argument is not required
Example :

```
rec = spi.byte(55)
spi.byte(123)
```

### spi.string():

Write and receive a string of len chars.
var$ = spi.string({string}, {len})
Example:

```
dat$ = spi.string("Hello World!", 12)
```

## spi.hex():

write and receive a string containing an hexadecimal message representing len bytes.
var $ = spi.string({hex_string}, {len})
Example:
*dat$ = spi.hex("a1b2c3d4e5f6", 6) -> send 6 bytes (sequence of a1, b2, c3, d1, e5, f6) and receive in dat$*

## spi.setmode():

Allows for changing of spi modes on the fly after the spi setup function has been used. Only modes 0 and 1 are supported.
spi.setmode({MODE})

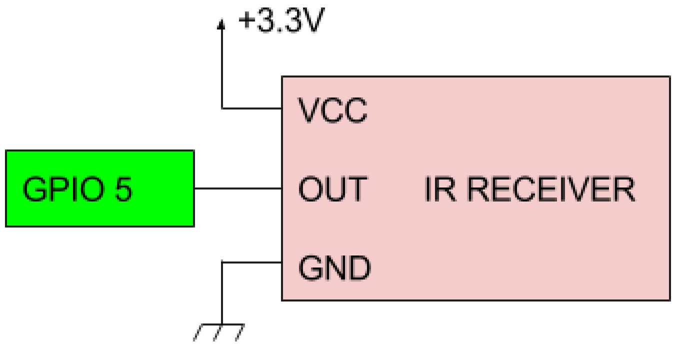
## spi.setfrequency():

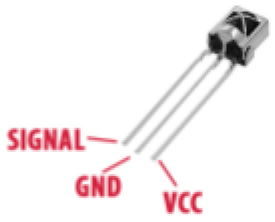
Allows for changing of spi frequency on the fly after the spi setup function has been used.
spi.setfrequency({FREQUENCY})

## FUNCTIONS IR:

### IR RECEIVE:

Several coding are supported; refer to the library for more details
The receiver can be a VS1838B or similar; this is very cheap (less than 2$ for 10!)
An hacked IR receiver can be found into a TV, old video recorder, ect....





### IRBRANCH:

Will specify a branch to execute when an infrared code is recognised. Branch must end with the return command.
irbranch [label]

### ir.recv.setup(pin)

set the pin used for the IR reception (IR receiver connected to the pin)
example:
*ir.recv.setup(5)      'set the pin 5*

### ir.recv.get()

get the code received ; the code is a string in hex format
Example:
*let r = ir.recv.get()      ->4907d8c5*

## ir.recv.full()

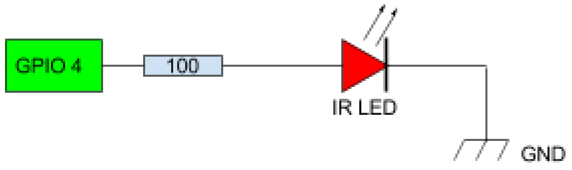
get the code received + the format + nb_of bits
this is useful to identify the kind of remote controller
and the size of the message
example:
*let r = ir.recv.full()     -> 4907d8c5:NEC:32*

## ir.recv.dump()

get the code received but verbose the message received on the serial port1 Probably more useful during debug
example:
*let r = ir.recv.dump()*

## IR TRANSMIT:

For the moment only the Sony and the NEX coding are supported for the transmission.



### ir.send.setup(pin)

set the pin used for the IR transmission (led connected to GND via a resistor)
example :
*ir.send.setup(4)     'set the pin 4*

### ir.send.nec(code, nb_bits)

send the code (string) composed of nb_bits bits
example:
*ir.send.nec("4907d8c5", 32)*

### ir.send.sony(code, nb_bits)

send the code (string) composed of nb_bits bits
example:
*ir.send.sony("910", 12)*

# FUNCTIONS UDP:

## Udpbegin:

Starts an UDP server listening on the port specified.
udpbegin {localport}
localport is a number (ex: 1234)
Example: *udpbegin 4300*

## Udpstop:

Stop the UDP server
udpstop
Works for UdpBegin and UdpBeginMulticast

## UdpbeginMultiCast:

Starts an UDP server multicast  listening at the address and the port specified.
udpbeginmulticast {multicast_ip}, {localport}
The multicast IP shall be in the range  224.0.0.0 to 239.255.255.255
Example : *udpbeginmulticast "224.0.0.67", 4567*

## Udpwrite:

Write a string toward a remote IP and  port
udpwrite {ip_address}, {port}, {message}
ip_address is a string (ex: "192.168.1.123")
message is a string (ex: "This is a valid message")
port is a number (ex: 5000)
Example: *udpwrite "192.168.1.12", 3000, "Message OK"*

## UdpwriteMultiCast:

Write a string toward a multicast IP and port
udpwritemulticast {ip_address}, {port}, {message}
ip_address is a string (ex: "224.0.0.50")
message is a string (ex: "This is a valid message")
port is a number (ex: 6000)
Example: *udpwritemulticast "224.0.0.55", 3100, "Message NOT delivered!"*

## UdpReply:

Send an UDP message back to the original transmitter.
Permit to answer directly without specify the IP and port
udpreply {message}
message is a string (ex: "This is a valid message")
Example: *udpreply "Message received"*

## UdpRead():

Read an UDP message received.
To works, the UDP server needs to be open (commands UdpBegin or UdpBeginMultiCast)
udpread()
Example: *let rec = udpread()*

## UdpRemote():

Read the IP address and the port of the message received
To works, the UDP server needs to be open (commands UdpBegin or UdpBeginMultiCast)
udpremote()
Example: let rem = udpremote()
The result will be in the form of IP:port; example (192.200.150.7:4568)

## UdpBranch:

Define a branch label defining the place where the program will continue as soon as an UDP message is received. As soon as the return command is found, the program will continue from the interrupted point.
To works, the UDP server needs to be open (commands UdpBegin or udpBeginMultiCast)
udpBranch {label}
*Example:*

```
udpbegin 4567
udpbranch [udp.received]
wait
[udp.received]
let rec = udpread()
let rem = udpremote()
Serialprint "Message received " & rec & " from " & rem
udpreply "OK"
return
```

## SPECIFICITY OF THE NEW PARSER

String Literals : it’s possible to use the " or the | as string literals;
This permits to include the " or | inside a string; example

```
A = |this is a "string"|
B = "this is a |string|"
```

Comparison operators for string and numbers:
The ‘=’ or ‘==’ gives the same result for string and numbers
The string can now be compared also for >, <, etc.

Boolan binary operators
The classic "basic" binary operators are available :
and, or, xor, not
Example: print 255 and 15;  let a = 255 xor 15; let b = not 15; let c = a or b

Shift operators
The shift left ‘<<’ and shift right ‘>>’ operators are available.
Example: print 15 << 3;      let c = a >> 4

Modulo Operator (%)
The modulo operator ‘%’ is available
Example: print 10 % 3

Boolean operators in ‘if’
All the boolean operators can be used in the ‘if’ command :
If a = b and c<>d then …..

All the comparisons can be used into expressions. If the result is true, the value will be -1, 0 is false
Example

```
let a = 5 <> 3  ⇒ -1
print 5+3 = 8   ⇒ -1
print 5-4 = 2   ⇒ 0
```

Spaces are don’t care :
A = 5 + 3 is the same as a=5+3

# ASCII TABLE OF CHARACTERS

| DEC | HEX | Name | Description |
|---|---|---|---|
| 0 | 00 | NUL | Null char |
| 1 | 01 | SOH | Start of Heading |
| 2 | 02 | STX | Start of Text |
| 3 | 03 | ETX | End of Text |
| 4 | 04 | EOT | End of Transmission |
| 5 | 05 | ENQ | Enquiry |
| 6 | 06 | ACK | Acknowledgment |
| 7 | 07 | BEL | Bell |
| 8 | 08 | BS | Back Space |
| 9 | 09 | HT | Horizontal Tab |
| 10 | 0A | LF | Line Feed |
| 11 | 0B | VT | Vertical Tab |
| 12 | 0C | FF | Form Feed |
| 13 | 0D | CR | Carriage Return |
| 14 | 0E | SO | Shift Out / X-On |
| 15 | 0F | SI | Shift In / X-Off |
| 16 | 10 | DLE | Data Line Escape |
| 17 | 11 | DC1 | Device Control 1 (oft. XON) |
| 18 | 12 | DC2 | Device Control 2 |
| 19 | 13 | DC3 | Device Control 3 (oft. XOFF) |
| 20 | 14 | DC4 | Device Control 4 |
| 21 | 15 | NAK | Negative Acknowledgement |
| 22 | 16 | SYN | Synchronous Idle |
| 23 | 17 | ETB | End of Transmit Block |
| 24 | 18 | CAN | Cancel |
| 25 | 19 | EM | End of Medium |
| 26 | 1A | SUB | Substitute |
| 27 | 1B | ESC | Escape |
| 28 | 1C | FS | File Separator |
| 29 | 1D | GS | Group Separator |
| 30 | 1E | RS | Record Separator |
| 31 | 1F | US | Unit Separator |
| 32 | 20 |  | Space |
| 33 | 21 | ! | Exclamation mark |
| 34 | 22 | " | &quot; Double quotes (or speech marks) |
| 35 | 23 | # | Number |
| 36 | 24 | $ | Dollar |
| 37 | 25 | % | Procenttecken |
| 38 | 26 | & | Ampersand |
| 39 | 27 | ' | Single quote |

| | | | |
|---|---|---|---|
| 40 | 28 | ( | Open parenthesis (or open bracket) |
| 41 | 29 | ) | Close parenthesis (or close bracket) |
| 42 | 2A | * | Asterisk |
| 43 | 2B | + | Plus |
| 44 | 2C | , | Comma |
| 45 | 2D | - | Hyphen |
| 46 | 2E | . | Period, dot or full stop |
| 47 | 2F | / | Slash or divide |
| 48 | 30 | 0 | Zero |
| 49 | 31 | 1 | One |
| 50 | 32 | 2 | Two |
| 51 | 33 | 3 | Three |
| 52 | 34 | 4 | Four |
| 53 | 35 | 5 | Five |
| 54 | 36 | 6 | Six |
| 55 | 37 | 7 | Seven |
| 56 | 38 | 8 | Eight |
| 57 | 39 | 9 | Nine |
| 58 | 3A | : | Colon |
| 59 | 3B | ; | Semicolon |
| 60 | 3C | < | Less than (or open angled bracket) |
| 61 | 3D | = | Equals |
| 62 | 3E | > | Greater than (or close angled bracket) |
| 63 | 3F | ? | Question mark |
| 64 | 40 | @ | At symbol |
| 65 | 41 | A | Uppercase A |
| 66 | 42 | B | Uppercase B |
| 67 | 43 | C | Uppercase C |
| 68 | 44 | D | Uppercase D |
| 69 | 45 | E | Uppercase E |
| 70 | 46 | F | Uppercase F |
| 71 | 47 | G | Uppercase G |
| 72 | 48 | H | Uppercase H |
| 73 | 49 | I | Uppercase I |
| 74 | 4A | J | Uppercase J |
| 75 | 4B | K | Uppercase K |
| 76 | 4C | L | Uppercase L |
| 77 | 4D | M | Uppercase M |
| 78 | 4E | N | Uppercase N |
| 79 | 4F | O | Uppercase O |
| 80 | 50 | P | Uppercase P |
| 81 | 51 | Q | Uppercase Q |
| 82 | 52 | R | Uppercase R |
| 83 | 53 | S | Uppercase S |
| 84 | 54 | T | Uppercase T |
| 85 | 55 | U | Uppercase U |

| | | | |
|---|---|---|---|
| 86 | 56 | V | Uppercase V |
| 87 | 57 | W | Uppercase W |
| 88 | 58 | X | Uppercase X |
| 89 | 59 | Y | Uppercase Y |
| 90 | 5A | Z | Uppercase Z |
| 91 | 5B | [ | Opening bracket |
| 92 | 5C | \ | Backslash |
| 93 | 5D | ] | Closing bracket |
| 94 | 5E | ^ | Caret - circumflex |
| 95 | 5F | _ | Underscore |
| 96 | 60 | ` | Grave accent |
| 97 | 61 | a | Lowercase a |
| 98 | 62 | b | Lowercase b |
| 99 | 63 | c | Lowercase c |
| 100 | 64 | d | Lowercase d |
| 101 | 65 | e | Lowercase e |
| 102 | 66 | f | Lowercase f |
| 103 | 67 | g | Lowercase g |
| 104 | 68 | h | Lowercase h |
| 105 | 69 | i | Lowercase i |
| 106 | 6A | j | Lowercase j |
| 107 | 6B | k | Lowercase k |
| 108 | 6C | l | Lowercase l |
| 109 | 6D | m | Lowercase m |
| 110 | 6E | n | Lowercase n |
| 111 | 6F | o | Lowercase o |
| 112 | 70 | p | Lowercase p |
| 113 | 71 | q | Lowercase q |
| 114 | 72 | r | Lowercase r |
| 115 | 73 | s | Lowercase s |
| 116 | 74 | t | Lowercase t |
| 117 | 75 | u | Lowercase u |
| 118 | 76 | v | Lowercase v |
| 119 | 77 | w | Lowercase w |
| 120 | 78 | x | Lowercase x |
| 121 | 79 | y | Lowercase y |
| 122 | 7A | z | Lowercase z |
| 123 | 7B | { | Opening brace |
| 124 | 7C | \| | Vertical bar |
| 125 | 7D | } | Closing brace |
| 126 | 7E | ~ | Equivalency sign - tilde |
| 127 | 7F | | Delete |
| 128 | 80 | € | &euro;    Euro sign |
| 129 | 81 | | |
| 130 | 82 | ‚ | Single low-9 quotation mark |
| 131 | 83 | ƒ | Latin small letter f with hook |

| | | | |
|---|---|---|---|
| 132 | 84 | „ | Double low-9 quotation mark |
| 133 | 85 | … | Horizontal ellipsis |
| 134 | 86 | † | Dagger |
| 135 | 87 | ‡ | Double dagger |
| 136 | 88 | ˆ | Modifier letter circumflex accent |
| 137 | 89 | ‰ | Per mille sign |
| 138 | 8A | Š | Latin capital letter S with caron |
| 139 | 8B | ‹ | Single left-pointing angle quotation |
| 140 | 8C | Œ | Latin capital ligature OE |
| 141 | 8D | | |
| 142 | 8E | Ž | Latin captial letter Z with caron |
| 143 | 8F | | |
| 144 | 90 | | |
| 145 | 91 | ‘ | Left single quotation mark |
| 146 | 92 | ’ | Right single quotation mark |
| 147 | 93 | " | Left double quotation mark |
| 148 | 94 | " | Right double quotation mark |
| 149 | 95 | • | Bullet |
| 150 | 96 | – | En dash |
| 151 | 97 | — | Em dash |
| 152 | 98 | | Small tilde |
| 153 | 99 | ™ | Trade mark sign |
| 154 | 9A | š | Latin small letter S with caron |
| 155 | 9B | › | Single right-pointing angle quotation mark |
| 156 | 9C | œ | Latin small ligature oe |
| 157 | 9D | | |
| 158 | 9E | ž | Latin small letter z with caron |
| 159 | 9F | Ÿ | Latin capital letter Y with diaeresis |
| 160 | A0 | | Non-breaking space |
| 161 | A1 | ¡ | Inverted exclamation mark |
| 162 | A2 | ¢ | Cent sign |
| 163 | A3 | £ | Pound sign |
| 164 | A4 | ¤ | Currency sign |
| 165 | A5 | ¥ | Yen sign |
| 166 | A6 | ¦ | Pipe, Broken vertical bar |
| 167 | A7 | § | Section sign |
| 168 | A8 | ¨ | Spacing diaeresis - umlaut |
| 169 | A9 | © | Copyright sign |
| 170 | AA | ª | Feminine ordinal indicator |
| 171 | AB | « | Left double angle quotes |
| 172 | AC | ¬ | Not sign |
| 173 | AD | | Soft hyphen |
| 174 | AE | ® | Registered trade mark sign |
| 175 | AF | ¯ | Spacing macron - overline |
| 176 | B0 | ° | Degree sign |
| 177 | B1 | ± | Plus-or-minus sign |

| | | | |
|---|---|---|---|
| 178 | B2 | ² | Superscript two - squared |
| 179 | B3 | ³ | Superscript three - cubed |
| 180 | B4 | ´ | Acute accent - spacing acute |
| 181 | B5 | µ | Micro sign |
| 182 | B6 | ¶ | Pilcrow sign - paragraph sign |
| 183 | B7 | · | Middle dot - Georgian comma |
| 184 | B8 | ¸ | Spacing cedilla |
| 185 | B9 | ¹ | Superscript one |
| 186 | BA | º | Masculine ordinal indicator |
| 187 | BB | » | Right double angle quotes |
| 188 | BC | ¼ | Fraction one quarter |
| 189 | BD | ½ | Fraction one half |
| 190 | BE | ¾ | Fraction three quarters |
| 191 | BF | ¿ | Inverted question mark |
| 192 | C0 | À | Latin capital letter A with grave |
| 193 | C1 | Á | Latin capital letter A with acute |
| 194 | C2 | Â | Latin capital letter A with circumflex |
| 195 | C3 | Ã | Latin capital letter A with tilde |
| 196 | C4 | Ä | Latin capital letter A with diaeresis |
| 197 | C5 | Å | Latin capital letter A with ring above |
| 198 | C6 | Æ | Latin capital letter AE |
| 199 | C7 | Ç | Latin capital letter C with cedilla |
| 200 | C8 | È | Latin capital letter E with grave |
| 201 | C9 | É | Latin capital letter E with acute |
| 202 | CA | Ê | Latin capital letter E with circumflex |
| 203 | CB | Ë | Latin capital letter E with diaeresis |
| 204 | CC | Ì | Latin capital letter I with grave |
| 205 | CD | Í | Latin capital letter I with acute |
| 206 | CE | Î | Latin capital letter I with circumflex |
| 207 | CF | Ï | Latin capital letter I with diaeresis |
| 208 | D0 | Ð | Latin capital letter ETH |
| 209 | D1 | Ñ | Latin capital letter N with tilde |
| 210 | D2 | Ò | Latin capital letter O with grave |
| 211 | D3 | Ó | Latin capital letter O with acute |
| 212 | D4 | Ô | Latin capital letter O with circumflex |
| 213 | D5 | Õ | Latin capital letter O with tilde |
| 214 | D6 | Ö | Latin capital letter O with diaeresis |
| 215 | D7 | × | Multiplication sign |
| 216 | D8 | Ø | Latin capital letter O with slash |
| 217 | D9 | Ù | Latin capital letter U with grave |
| 218 | DA | Ú | Latin capital letter U with acute |
| 219 | DB | Û | Latin capital letter U with circumflex |
| 220 | DC | Ü | Latin capital letter U with diaeresis |
| 221 | DD | Ý | Latin capital letter Y with acute |
| 222 | DE | Þ | Latin capital letter THORN |
| 223 | DF | ß | Latin small letter sharp s - ess-zed |

| | | | |
|---|---|---|---|
| 224 | E0 | à | Latin small letter a with grave |
| 225 | E1 | á | Latin small letter a with acute |
| 226 | E2 | â | Latin small letter a with circumflex |
| 227 | E3 | ã | Latin small letter a with tilde |
| 228 | E4 | ä | Latin small letter a with diaeresis |
| 229 | E5 | å | Latin small letter a with ring above |
| 230 | E6 | æ | Latin small letter ae |
| 231 | E7 | ç | Latin small letter c with cedilla |
| 232 | E8 | è | Latin small letter e with grave |
| 233 | E9 | é | Latin small letter e with acute |
| 234 | EA | ê | Latin small letter e with circumflex |
| 235 | EB | ë | Latin small letter e with diaeresis |
| 236 | EC | ì | Latin small letter i with grave |
| 237 | ED | í | Latin small letter i with acute |
| 238 | EE | î | Latin small letter i with circumflex |
| 239 | EF | ï | Latin small letter i with diaeresis |
| 240 | F0 | ð | Latin small letter eth |
| 241 | F1 | ñ | Latin small letter n with tilde |
| 242 | F2 | ò | Latin small letter o with grave |
| 243 | F3 | ó | Latin small letter o with acute |
| 244 | F4 | ô | Latin small letter o with circumflex |
| 245 | F5 | õ | Latin small letter o with tilde |
| 246 | F6 | ö | Latin small letter o with diaeresis |
| 247 | F7 | ÷ | Division sign |
| 248 | F8 | ø | Latin small letter o with slash |
| 249 | F9 | ù | Latin small letter u with grave |
| 250 | FA | ú | Latin small letter u with acute |
| 251 | FB | û | Latin small letter u with circumflex |
| 252 | FC | ü | Latin small letter u with diaeresis |
| 253 | FD | ý | Latin small letter y with acute |
| 254 | FE | þ | Latin small letter thorn |
| 255 | FF | ÿ | Latin small letter y with diaeresis |