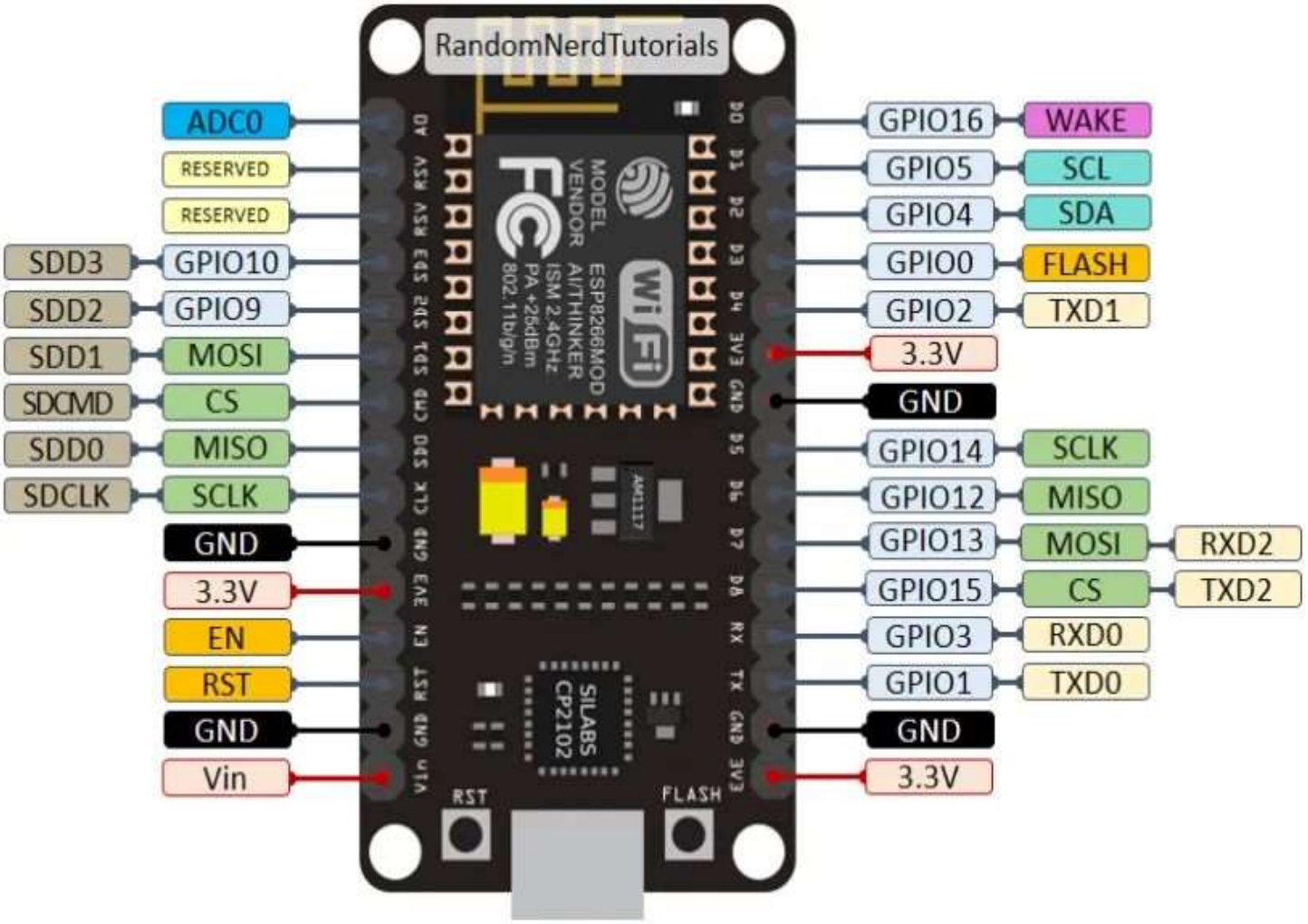
ESP8266 chip**Technical specifications :** The primary reference for the chip technical specifications, capabilities, operating modes, internal functioning,

- Architecture: Xtensa lx106
- CPU frequency: 80MHz overclockable to 160MHz
- Total RAM available: 96KB (part of it reserved for system)
- BootROM: 64KB
- Internal FlashROM: None
- External FlashROM: code and data, via SPI Flash. Normal sizes 512KB-4MB.
- GPIO: 16 + 1 (GPIOs are multiplexed with other functions, including external FlashROM, UART, deep sleep wake-up, etc.)
- UART: One RX/TX UART (no hardware handshaking), one TX-only UART.
- SPI: 2 SPI interfaces (one used for FlashROM).
- I2C: No native external I2C (bitbang implementation available on any pins).
- I2S: 1.
- Programming: using BootROM bootloader from UART. Due to external FlashROM and always available BootROM bootloader, ESP8266 is not brickable

Nodemcu 8266



Multitude of boards

- multitude of modules and boards from different sources which carry the ESP8266 chip.
- MicroPython tries to provide a generic port which would run on as many boards/modules
- Adafruit Feather HUZZAH board is taken as a reference
- If you have another board, please make sure you have a datasheet, schematics
- To make a generic ESP8266 port and support as many boards
- following design and implementation decision were made:1.GPIO pin numbering is based on ESP8266 chip numbering 2. All pins which make sense to support, are supported by MicroPython. 3. Some boards may lack external pins/internal connectivity to support ESP8266 deepsleep mode.

Scarcity of runtime resources

- ESP8266 has very modest resource(RAM)
- avoid allocating too big container objects (lists, dictionaries) and buffers
- There is also no full-fledged OS to keep track of resources and automatically clean them up
- so that’s the task of a user/user application (to close open files, sockets)

Boot process of Nodemcu 8266

- On boot Micropython ESP8266 executes _boot.py script from frozen modules
- It mounts the filesystem Into the **Fl**ash ROM or first setup of the module and creates filesystem
- Boot process is default process that runs on power up and is not available for customization for the user
- Once the filesystem is loaded then boot.py is executed
- Standard version of boot.py is created when the first time the module is setup (it contains commands to start WEBReply which is disabled by default. It is customizable to set up parameters and start services that need to start by default
- incorrect modifications to boot.py may still lead to boot loops or lock ups, requiring to reflash a module from scratch
- final step of boot procedure, main.py is executed from filesystem, if exists. This file is a hook to start up a user application each time on boot

Known Issues in Nodemcu 8266

- **Real-time clock** RTC in ESP8266 has very bad accuracy, drift may be seconds per minute. As a workaround, synchronize from the net using included ntptime.py module.
- **Sockets and WiFi buffers overflow** Socket instances remain active until they are explicitly closed. two consequences.

Firstly they occupy RAM, so an application which opens sockets without closing them may eventually run out of memory.

Secondly not properly closed socket can cause the low-level part of the vendor WiFi stack to emit Lmac errors.

- **SSL/TLS limitations** ESP8266 uses axTLS library, which is one of the smallest TLS libraries with the compatible licensing. However, it also has some known issues/limitations

1.No support for Diffie-Hellman (DH) key exchange and Elliptic-curve cryptography (ECC).

2.Half-duplex communication nature. axTLS uses a single buffer for both sending and receiving

axTLS own limitations, the configuration used for MicroPython is highly optimized for code

MicroPython language and implementation

- MicroPython aims to implement the Python 3.4 standard
- most of the features of MicroPython are identical
- “Language Reference” documentation at docs.python.org.

MicroPython language and implementation

- The MicroPython software supports the ESP8266 chip itself and any board should work
- main characteristic of a board is how much flash it has

  minimum requirement for flash size is 1Mbyte

how the GPIO pins are connected to the outside world

whether it includes a built-in USB-serial convertor to make the UART available to your PC.

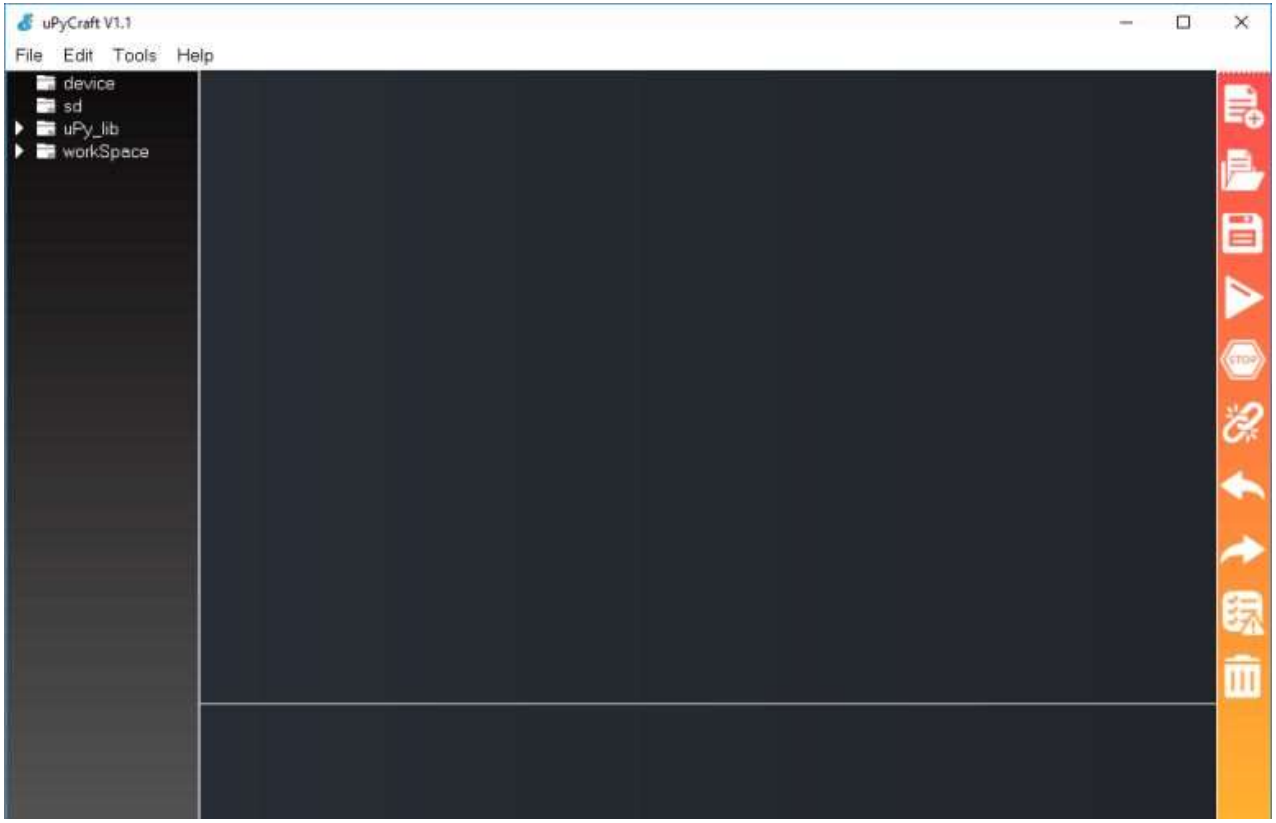
MicroPython firmware .bin file to load onto your ESP8266 device

3 main choices

- Stable firmware builds for 1024kb modules and above.
- Daily firmware builds for 1024kb modules and above.
- Daily firmware builds for 512kb modules.

Nodemcu 8266 Micropython IDE uPy

https://randomnerdtutorials.com/install-upycraft-ide-windows-pc-instructions/



MicroPython Interactive Interpreter Mode

- some characteristics of the MicroPython Interactive Interpreter Mode
- commonly used term for this is REPL (read-eval-print-loop)

>>> for i in

(30): ... _

Auto-indent

- When typing python statements which end in a **colon** (for example if, for, while) then the prompt will change to **three dots (...)** and the cursor will be indented by 4 spaces.

Eg 1.

```
>>> for i in range(30):
...     _
```

Eg2.

```
>>> for i in range(30):
...     if i > 3:
...         break
...     _
```

Auto-indent won't be applied if the previous two lines were all spaces.

- When you press return, the next line will continue at the same level of **indentation for regular statements or an additional level** of indentation where appropriate. If you press the backspace key then it will undo one level of indentation.

Auto-completion

- While typing a command at the REPL, if the line typed so far corresponds to the beginning of the name of something, then pressing TAB will show possible things that could be entered.

Eg

- first import the **machine module** by entering **import machine** and pressing RETURN. Then type m and press TAB and it should expand to machine. Enter a dot . and press TAB again. You should see something like:
- >>> machine.

__name__ info unique_id reset

bootloader freq rng idle

sleep deepsleep disable_irq enable_irq

Pin

Auto-completion

- The word will be expanded as much as possible until multiple possibilities exist. For example, type machine.Pin.AF3 and press TAB and it will expand to machine.Pin.AF3_TIM. Pressing TAB a second time will show the possible expansions:

- >> machine.Pin.AF3_TIM

AF3_TIM10 AF3_TIM11 AF3_TIM8 AF3_TIM9

- >>> machine.Pin.AF3_TIM

Interrupting a running program

You can interrupt a running program by pressing Ctrl-C. This will raise a KeyboardInterrupt which will bring you back to the REPL, providing your program doesn't intercept the KeyboardInterrupt exception.

For example:

```
>>> for i in range(1000000)
...     print(i)
...
0
1
...
6467
6468
Traceback (most recent call last):
 File "<stdin>", line 2, in <module>
KeyboardInterrupt:
>>>
```

Paste mode

if you want to paste some code into your terminal window, the auto-indent feature will mess things up. For example, if you had the following python code:

```
def foo():
    print('This is a test to show paste mode')
     print('Here is a second line')
foo()
```

Paste mode

and you try to paste this into the normal REPL, then you will see something like this:

```
>>> def foo():
...         print('This is a test to show paste mode')
...             print('Here is a second line')
...             foo()
...
  File "<stdin>", line 3
IndentationError: unexpected indent
```

Paste mode

- If you press Ctrl-E, then you will enter paste mode, which essentially turns off the auto-indent feature, and changes the prompt from >>> to ===. For example:

```
>>>
paste mode; Ctrl-C to cancel, Ctrl-D to finish
=== def foo():
===     print('This is a test to show paste mode')
===     print('Here is a second line')
=== foo()
===
This is a test to show paste mode
Here is a second line
>>>
```

Paste Mode allows blank lines to be pasted. The pasted text is compiled as if it were a file. Pressing Ctrl-D exits paste mode and initiates the compilation.

Soft reset

- A soft reset will reset the python interpreter, but tries not to reset the method by which you're connected to the MicroPython board (USB-serial, or Wifi).
- You can perform a soft reset from the REPL by pressing Ctrl-D, or from your python code by executing:
- machine.soft_reset()
- For example, if you reset your MicroPython board, and you execute a dir() command, you'd see something like this:

```
>>> dir()
['__name__', 'pyb']
```

Soft reset

Now create some variables and repeat the dir() command:

```
>>> i = 1
>>> j = 23
>>> x = 'abc'
>>> dir()
['j', 'x', '__name__', 'pyb', 'i']
>>>
```

Now if you enter Ctrl-D, and repeat the dir() command, you'll see that your variables no longer exist:

```
MPY: sync filesystems
MPY: soft reboot
MicroPython v1.5-51-g6f70283-dirty on 2015-10-30; PYBv1.0 with STM32F405RG
Type "help()" for more information.
>>> dir()
['__name__', 'pyb']
>>>
```