



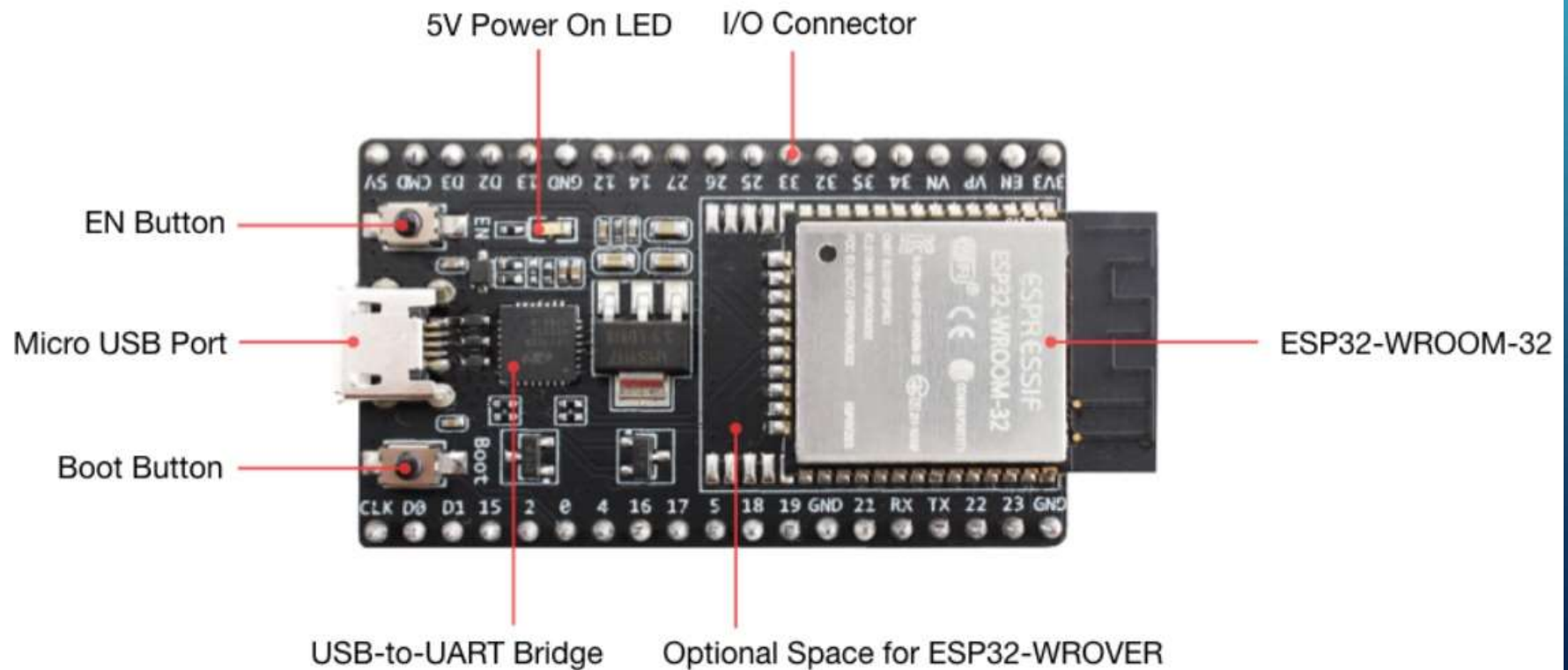
ZÁKLADY BEZDRÔTOVÝCH SIETÍ

KOMUNIKAČNÉ TECHNOLOGIE PRE SYSTÉMY IOT

ING. LUKÁŠ FORMANEK, PHD.

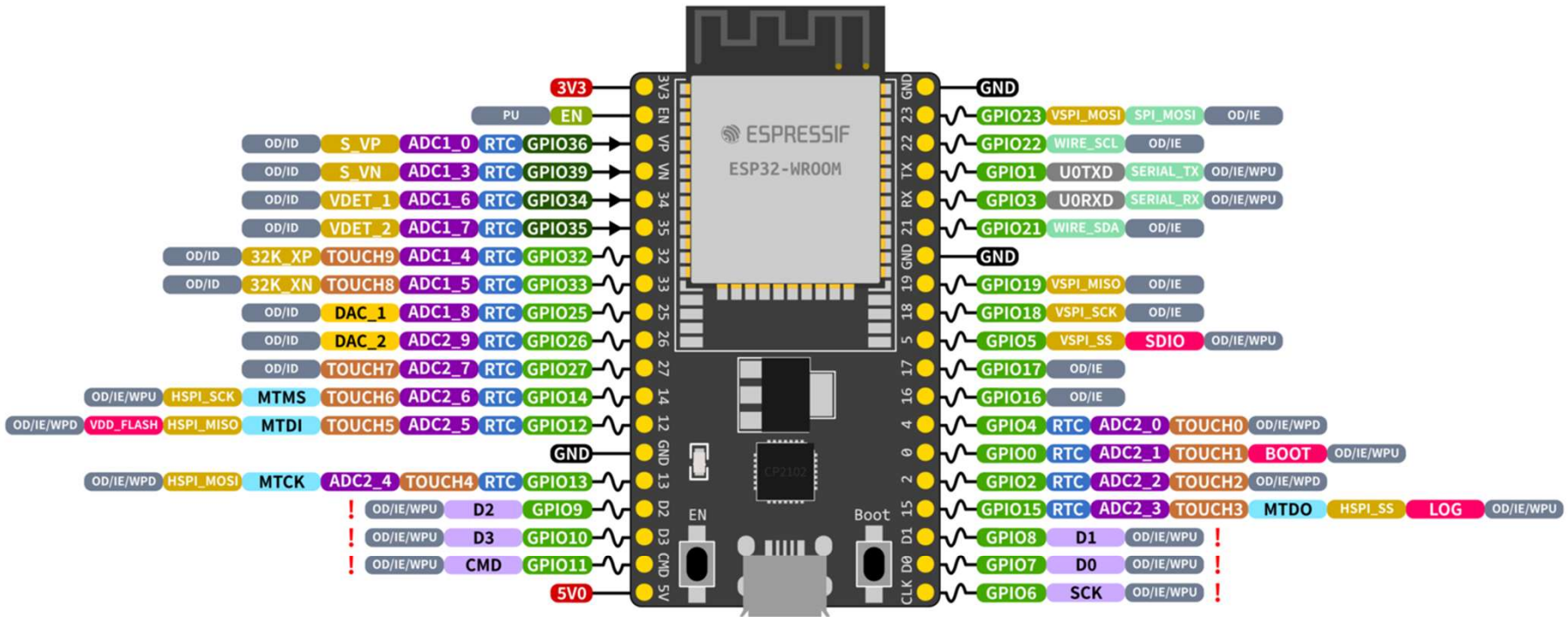
Vytvorené v rámci projektu KEGA 026TUKE-4/2021

ESP32-DEVKITC



* Používate na predmete : VYS.

ESP32-DevKitC



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
 Wi-Fi IEEE 802.11 b/g/n 2.4GHz
 Bluetooth 4.2 BR/EDR and BLE
 520 KB SRAM (16 KB for cache)
 448 KB ROM
 34 GPIOs, 4x SPI, 3x UART, 2x I2C,
 2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
 1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

- PWM Capable Pin
- GPIOX GPIO Input Only
- GPIOX GPIO Input and Output
- DAC_X Digital-to-Analog Converter
- DEBUG JTAG for Debugging
- FLASH External Flash Memory (SPI)
- ADCX_CH Analog-to-Digital Converter
- TOUCHX Touch Sensor Input Channel
- OTHER Other Related Functions
- SERIAL Serial for Debug/Programming
- ARDUINO Arduino Related Functions
- STRAP Strapping Pin Functions

- RTC RTC Power Domain (VDD3P3_RTC)
- GND Ground
- PWD Power Rails (3V3 and 5V)
- Pin Shared with the Flash Memory
Can't be used as regular GPIO

GPIO STATE

- WPU:** Weak Pull-up (Internal)
- WPD:** Weak Pull-down (Internal)
- PU:** Pull-up (External)
- IE:** Input Enable (After Reset)
- ID:** Input Disabled (After Reset)
- OE:** Output Enable (After Reset)
- OD:** Output Disabled (After Reset)

ESP32-DEVKITC

- <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html>
- Schematic: https://dl.espressif.com/dl/schematics/esp32_devkitc_v4-sch.pdf

ESP32 DEVELOPMENT BOARD

ESP32 DEV KIT V1 PINOUT

PWR LED (points to pin 1)

USER LED (GPIO2) (points to pin 2)

UART (points to pins 40, 41, 42)

RST (points to RST pin)

BOOT (GPIO0) (points to BOOT pin)

Pin	Function
EN	EN
5	GPIO36 (Input only), RTC GPIO0, SensVP, ADC1_0
8	GPIO39 (Input only), RTC GPIO3, SensVN, ADC1_3
10	GPIO34 (Input only), RTC GPIO4, ADC1_6
11	GPIO35 (Input only), RTC GPIO5, ADC1_7
12	GPIO32 (RTC GPIO9), Xtal32P, Touch9, ADC1_4
13	GPIO33 (RTC GPIO8), Xtal32N, Touch8, ADC1_5
14	GPIO25 (DAC 1), RTC GPIO6, ADC2_8
15	GPIO26 (DAC 2), RTC GPIO7, ADC2_9
16	GPIO27 (RTC GPIO17), Touch7, ADC2_7
17	GPIO14 (RTC GPIO16), Touch6, HSPI_CLK, ADC2_6
18	GPIO12 (RTC GPIO15), Touch5, HSPI_Q, ADC2_5
20	GPIO13 (RTC GPIO14), Touch4, HSPI_ID, ADC2_4
36	GPIO23, V_SPI_D, MOSI
39	GPIO22, V_SPI_WP, SCL, RTS 0
41	GPIO1 (TXD 0), CLK3
40	GPIO3 (RXD 0), CLK2
42	GPIO21, VSPI_HD, SDA
38	GPIO19, V_SPI_Q, MISO, CTS 0
35	GPIO18, V_SPI_CLK, SCK
34	GPIO5, V_SPI_CS0, SS
27	GPIO17, TXD 2
25	GPIO16, RXD 2
24	GPIO4, ADC2_0, HSPI_HD, Touch0, RTC GPIO10
22	GPIO2, ADC2_2, HSPI_WP0, Touch2, RTC GPIO12
21	GPIO15, ADC2_3, HSPI_CS0, Touch3, RTC GPIO13
GND	GND
3.3v	3.3v

www.mischianti.org

ESP32-WROOM-32 (DATASHEET)

- https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf



ESP32-WR00M-32

MICROPYTHON

- Micropython:

<https://micropython.org/>

- Firmware:

<https://micropython.org/download/esp32/>

- Docs:

<https://docs.micropython.org/en/latest/esp32/quickref.html>

Vendor: Espressif
Features: BLE, WiFi
Source on GitHub: [esp32/GENERIC](#)
More info: [Website](#)

The following files are daily firmware for ESP32-based boards without external SPIRAM.
This firmware is compiled using ESP-IDF v4.x. Some older releases are also provided that are compiled with ESP-IDF v3.x.

Installation instructions

Program your board using the `esptool.py` program, found [here](#).

If you are putting MicroPython on your board for the first time then you should first erase the entire flash using:

```
esptool.py --chip esp32 --port /dev/ttyUSB0 erase_flash
```

From then on program the firmware starting at address 0x1000:

```
esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 460800 write_flash -z 0x1000 esp32-20190125-v1.10.bin
```

Firmware

Releases

v1.18 (2022-01-17)	.bin	[elf]	[map]	[Release notes] (latest)
v1.17 (2021-09-02)	.bin	[elf]	[map]	[Release notes]
v1.16 (2021-06-23)	.bin	[elf]	[map]	[Release notes]
v1.15 (2021-04-18)	.bin	[elf]	[map]	[Release notes]
v1.14 (2021-02-02)	.bin	[elf]	[map]	[Release notes]
v1.13 (2020-09-02)	.bin	[elf]	[map]	[Release notes]
v1.12 (2019-12-20)	.bin	[elf]	[map]	[Release notes]

Nightly builds

v1.18-307-g66b5c4677 (2022-04-11)	.bin	[elf]	[map]	
v1.18-305-g567339022 (2022-04-07)	.bin	[elf]	[map]	
v1.18-304-g5682595d7 (2022-04-05)	.bin	[elf]	[map]	
v1.18-300-g3699b4d67 (2022-04-04)	.bin	[elf]	[map]	

Firmware (Compiled with IDF 3.x)

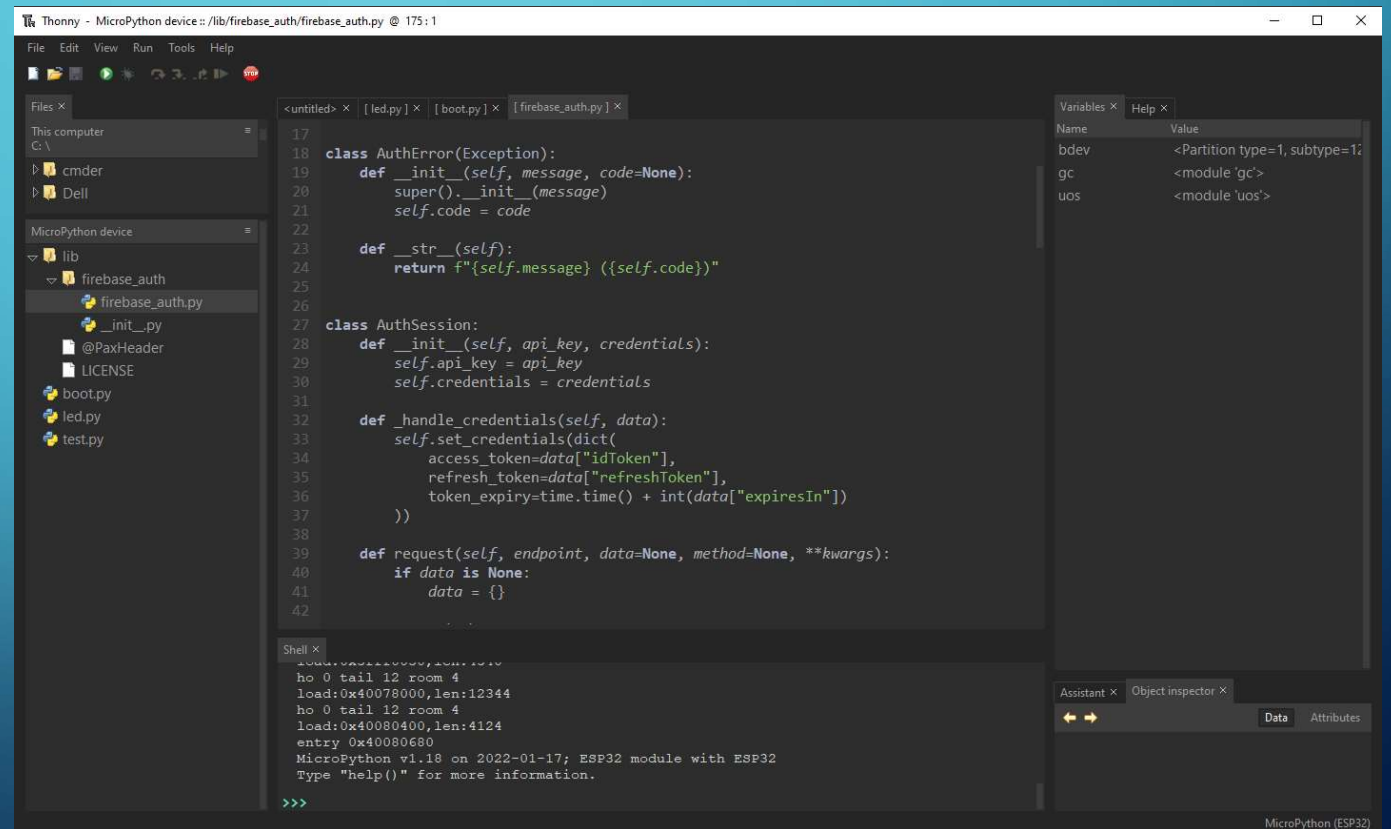
Releases

v1.14 (2021-02-02)	.bin	[elf]	[map]	[Release notes] (latest)
v1.13 (2020-09-02)	.bin	[elf]	[map]	[Release notes]
v1.12 (2019-12-20)	.bin	[elf]	[map]	[Release notes]
v1.11 (2019-05-29)	.bin	[elf]	[map]	[Release notes]
v1.10 (2019-01-25)	.bin	[elf]	[map]	[Release notes]
v1.9.4 (2018-05-11)	.bin	[elf]	[map]	[Release notes]

* 1.19.1 (latest)

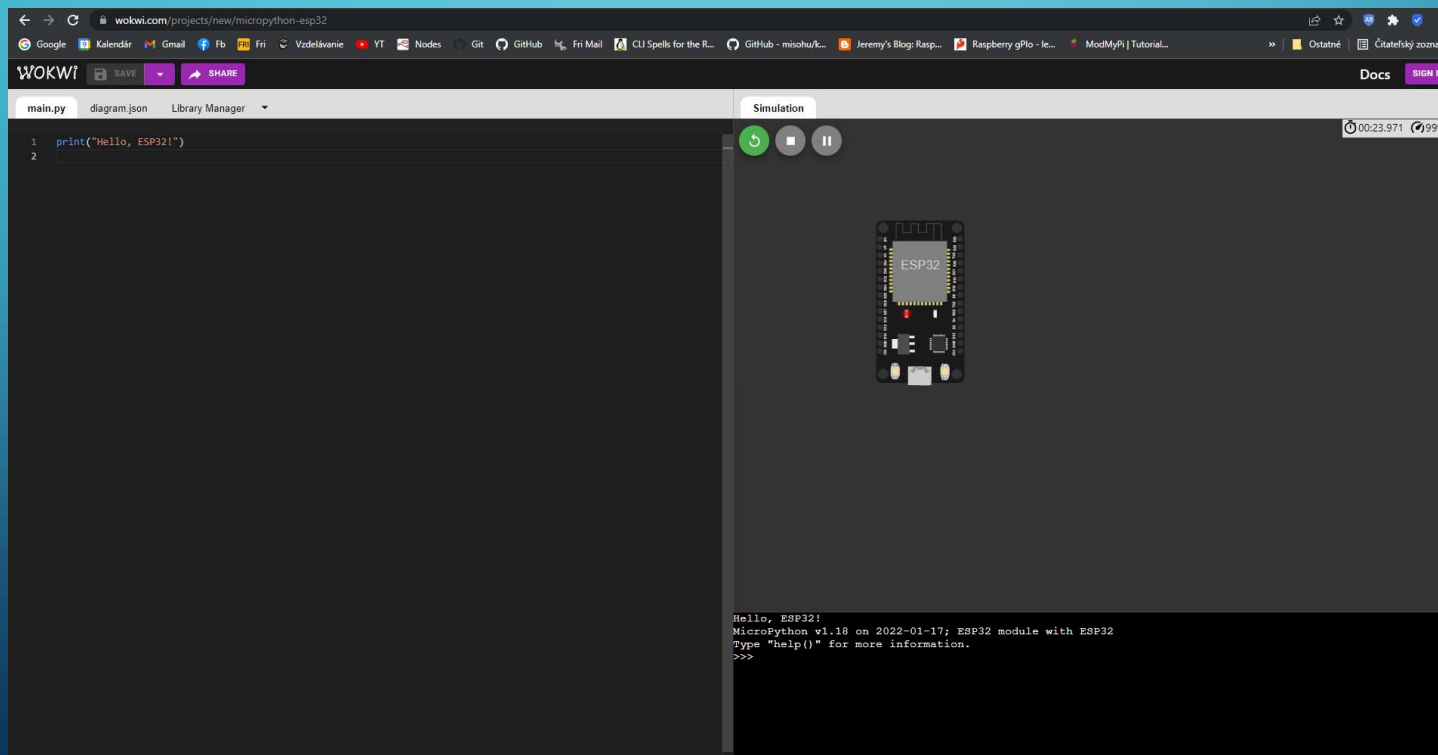
THONNY (IDE)

- <https://thonny.org/>



ESP32 + MICROPYTHON – SIMULATOR

- <https://wokwi.com/projects/new/micropython-esp32>



The screenshot displays the Wokwi web-based simulator interface. The browser address bar shows the URL `wokwi.com/projects/new/micropython-esp32`. The interface is split into two main sections:

- Code Editor:** On the left, a file named `main.py` is open. It contains two lines of Python code:

```
1 print("Hello, ESP32!")  
2
```
- Simulation Window:** On the right, a virtual ESP32 microcontroller board is shown. Below the board, a terminal window displays the output of the code:

```
Hello, ESP32!  
MicroPython v1.18 on 2022-01-17; ESP32 module with ESP32  
Type "help()" for more information.  
>>>
```

At the top of the simulation window, there are control buttons for running (a green play button), pausing (a square button), and stopping (a red stop button). A timer in the top right corner of the simulation window shows `00:23.971` and `99%` battery level.

FIRMWARE UPDATE

The screenshot illustrates the steps to update the firmware of an ESP32 device using Thonny. The process involves navigating through the 'Options' menu to the 'Thonny options' dialog, selecting the 'Interpreter' tab, and then opening the 'ESP32 firmware installer' dialog. In the 'ESP32 firmware installer' dialog, the 'Port' is set to 'Silicon Labs CP210x USB to UART Bridge (COM12)' and the 'Firmware' is set to 'C:/Users/lukas/Downloads/esp32-20220117-v1.18 (3).bin'. The 'Install' button is highlighted, indicating the final step in the process.

Thonny options dialog (Interpreter tab):

- Which interpreter or device should Thonny use for running your code?
MicroPython (ESP32)
- Port or WebREPL:
Silicon Labs CP210x USB to UART Bridge (COM12)
- Install or update firmware

ESP32 firmware installer dialog:

- Port: Silicon Labs CP210x USB to UART Bridge (COM12)
- Firmware: C:/Users/lukas/Downloads/esp32-20220117-v1.18 (3).bin
- Flash mode:
 From image file (keep) Quad I/O (qio)
 Dual I/O (dio) Dual Output (dout)
 Erase flash before installing
- Buttons: Install, Cancel

Shell output:

```
load:0x40080000,len:1234
entry 0x40080000
MicroPython v1.18 on 2022-01-17; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

IDE SETTINGS

The screenshot displays the Thonny IDE interface for a MicroPython device. The main window title is "Thonny - MicroPython device :: /boot.py @ 6:1". The menu bar includes File, Edit, View, Run, Tools, and Help. The View menu is open, showing options like Assistant, Exception, Files, Heap, Help, Notes, Object inspector, Outline, Program tree, Shell, Stack, and Variables. Below these are Program arguments and Plotter, followed by font size controls (Increase font size Ctrl++ and Decrease font size Ctrl+-) and focus controls (Focus editor Alt+E and Focus shell Alt+S). The code editor shows the following code:

```
[boot.py] x
1 # This file is executed on every boot (including wake-boot from deepsleep)
2 #import esp
3 #esp.osdebug(None)
4 #import webrepl
5 #webrepl.start()
6
```

The Shell window at the bottom shows the following output:

```
Shell x
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

MicroPython v1.18 on 2022-01-17; ESP32 module with ESP32
Type "help()" for more information.
MicroPython v1.18 on 2022-01-17; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

On the right side, the Variables window shows the following table:

Name	Value
bdev	<Partition type=1, subtype=12>
gc	<module 'gc'>
uos	<module 'uos'>

Below the Variables window is the Object inspector window, which shows the following table:

Name	Value
VfsFat	<class 'VfsFat'>
VfsLfs2	<class 'VfsLfs2'>
chdir	<function>
dupterm	<function>
dupterm_notify	<function>
getcwd	<function>
ilistdir	<function>
listdir	<function>
mkdir	<function>
mount	<function>
remove	<function>
rename	<function>
rmdir	<function>
stat	<function>
statvfs	<function>
umount	<function>
uname	<function>
urandom	<function>

IDE

Run script

Stop/Restart

Files

MicroPython device :: /boot.py @ 6:1

File Edit View Run Tools Help

Files ×

- This computer
- C:\
- cmdr
- Dell
- MicroPython device
- boot.py

Stop/Restart backend (Ctrl+F2)

```
1 # This file is executed on every boot (including wake-boot from deepsleep)
2 #import esp
3 #esp.osdebug(None)
```

Shell ×

```
>>> help()
Welcome to MicroPython on the ESP32!

For generic online docs please visit http://docs.micropython.org/

For access to the hardware use the 'machine' module:

import machine
pin12 = machine.Pin(12, machine.Pin.OUT)
pin12.value(1)
pin13 = machine.Pin(13, machine.Pin.IN, machine.Pin.PULL_UP)
print(pin13.value())
i2c = machine.I2C(scl=machine.Pin(21), sda=machine.Pin(22))
i2c.scan()
i2c.writeto(addr, b'1234')
i2c.readfrom(addr, 4)

Basic WiFi configuration:

import network
sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan() # Scan for available access points
sta_if.connect("<AP_name>", "<password>") # Connect to an AP
sta_if.isconnected() # Check for successful connection

Control commands:
CTRL-A -- on a blank line, enter raw REPL mode
CTRL-B -- on a blank line, enter normal REPL mode
CTRL-C -- interrupt a running program
CTRL-D -- on a blank line, do a soft reset of the board
CTRL-E -- on a blank line, enter paste mode

For further help on a specific object, type help(obj)
For a list of available modules, type help('modules')

>>> |
```

Variables × Help ×

Name	Value
bdev	<Partition type=1, subtype=12>
gc	<module 'gc'>
uos	<module 'uos'>

Assistant × Object inspector ×

Name	Value
VfsFat	<class 'VfsFat'>
VfsLfs2	<class 'VfsLfs2'>
chdir	<function>
dupterm	<function>
dupterm_notify	<function>
getcwd	<function>
ilistdir	<function>
listdir	<function>
makedirs	<function>
mount	<function>
remove	<function>
rename	<function>
rmdir	<function>
stat	<function>
statvfs	<function>
umount	<function>
uname	<function>
urandom	<function>

MicroPython (ESP32)

REPL (Read Evaluate Print Loop) prompt

MICROPYTHON BUILT-IN MODULES (LIBRARIES)

The screenshot shows the Thonny IDE interface for a MicroPython device. The main editor displays the contents of `boot.py`, which includes comments and code for connecting to a Wi-Fi network. A red box highlights the first two lines of the code:

```
1 # This file is executed on every boot (including wake-boot from deepsleep)
2 #import esp
   #esp.osdebug(None)
```

The Shell window shows the execution of `import network` and the output of `help('modules')`, which lists the following built-in modules:

```
>>> help('modules')
__main__      gc              ubinascii      upysh
__boot       inisetup       ubluetooth     urandom
__onewire    machine        ucollections   ure
__thread     math           ucryptolib    urequests
_uasyncio    micropython    ctypes         uselect
__webrepl    neopixel       ueerrno        usecket
apa106       network        uhashlib       ussl
btree        ntptime        uheapq         ustruct
builtins     onewire        uio            usys
cmath        uarray         ujson          utime
dht          uasyncio/___init___  umqtt/robust  utimeq
ds18x20      uasyncio/core  umqtt/simple  uwebsocket
esp          uasyncio/event uos            uzlib
esp32       uasyncio/funcs upip           webrepl
flashbdev   uasyncio/lock  upip_utarfile webrepl_setup
framebuf    uasyncio/stream  uplatform     websocket_helper
Plus any modules on the filesystem

>>>
```

The Variables window on the right shows the current state of variables:

Name	Value
bdev	<Partition type=1, subtype=12>
gc	<module 'gc'>
uos	<module 'uos'>

The Object inspector window shows the current object being inspected:

Name	Value
VfsFat	<class 'VfsFat'>
VfsLfs2	<class 'VfsLfs2'>
chdir	<function>
dupterm	<function>
dupterm_notify	<function>
getcwd	<function>
ilistdir	<function>
listdir	<function>
makedirs	<function>
mount	<function>
remove	<function>
rename	<function>
rmdir	<function>
stat	<function>
statvfs	<function>
umount	<function>
uname	<function>
urandom	<function>

POUŽITIE KNIŽNICE (REPL)

The screenshot shows the Thonny IDE interface for a MicroPython device. The main window displays the REPL with the following content:

```
1 # This file is executed on every boot (including wake-boot from deepsleep)
2 #import esp
3 #esp.osdebug(None)

sta_if.isconnected() # Check for successful connection

Control commands:
CTRL-A -- on a blank line, enter raw REPL mode
CTRL-B -- on a blank line, enter normal REPL mode
CTRL-C -- interrupt a running program
CTRL-D -- on a blank line, do a soft reset of the board
CTRL-E -- on a blank line, enter paste mode

For further help on a specific object, type help(obj)
For a list of available modules, type help('modules')

>>> help('modules')
__main__      gc              ubinascii      upysh
_boot         inisetup       ubluetooth     urandom
_owewire     machine        ucollections   ure
_thread      math           ucryptolib    urequests
_uasyncio    micropython   uctypes       uselect
_webrepl     neopixel      uerrno        usocket
apa106       network       uhashlib      ussl
btree        ntptime       uheapq        ustruct
builtins     onewire       uio           usys
cmath        uarray        ujson         utime
dht          uasyncio/_init_ umqtt/robust  utimeq
ds18x20      uasyncio/core  umqtt/simple  uwebsocket
esp          uasyncio/event uos           uzlib
esp32        uasyncio/funcs upip          webrepl
flashbdev   uasyncio/lock  upip_utarfile webrepl_setup
framebuf    uasyncio/stream  uplatform    websocket_helper
Plus any modules on the filesystem

>>> import uos
>>> uos.listdir()
['boot.py']
>>>
```

The REPL output shows the result of `uos.listdir()` as `['boot.py']`. The interface also features several panels:

- Files:** Shows the file explorer for the device, with `boot.py` selected.
- Variables:** A table showing the current state of variables in the REPL. The `uos` variable is highlighted with a red box, showing its value as `<module 'uos'>`.
- Object Inspector:** A panel titled "Object inspector" showing the attributes of the `uos` module. It lists various methods and classes, with the entire panel highlighted by a red box.

Name	Value
__main__	gc
_boot	inisetup
_owewire	machine
_thread	math
_uasyncio	micropython
_webrepl	neopixel
apa106	network
btree	ntptime
builtins	onewire
cmath	uarray
dht	uasyncio/_init_
ds18x20	uasyncio/core
esp	uasyncio/event
esp32	uasyncio/funcs
flashbdev	uasyncio/lock
framebuf	uasyncio/stream
gc	ubinascii
ubluetooth	ubluetooth
ucollections	ucollections
ucryptolib	ucryptolib
uctypes	uctypes
uerrno	uerrno
uhashlib	uhashlib
uheapq	uheapq
uio	uio
ujson	ujson
umqtt/robust	umqtt/robust
umqtt/simple	umqtt/simple
uos	uos
upip	upip
upip_utarfile	upip_utarfile
uplatform	uplatform
upysh	upysh
urandom	urandom
urequests	urequests
uselect	uselect
usocket	usocket
ussl	ussl
ustruct	ustruct
usys	usys
utime	utime
utimeq	utimeq
uwebsocket	uwebsocket
uzlib	uzlib
webrepl	webrepl
webrepl_setup	webrepl_setup
websocket_helper	websocket_helper
VfsFat	<class 'VfsFat'>
VfsLfs2	<class 'VfsLfs2'>
chdir	<function>
dupterm	<function>
dupterm_notify	<function>
getcwd	<function>
ilistdir	<function>
listdir	<function>
mkdir	<function>
mount	<function>
remove	<function>
rename	<function>
rmdir	<function>
stat	<function>
statvfs	<function>
umount	<function>
uname	<function>
urandom	<function>

VYTVORENIE SCRIPTU

The screenshot displays the MicroPython IDE interface. The main editor window shows a Python script with the following code:

```
1 import uos
2
3 print(uos.)
```

A red box highlights the code completion menu that appears after typing `uos.`. The menu lists several methods: `dupterm_notify`, `getcwd`, `ilistdir`, `listdir` (highlighted in blue), and `mkdir`. A red arrow points from the text **CTRL+SPACE** to the `listdir` option in the menu.

The right-hand side of the IDE features a **Variables** panel and an **Object inspector**. The **Variables** panel shows the following:

Name	Value
_	['boot.py']
bdev	<Partition type=1, subtype=12>
gc	<module 'gc'>
uos	<module 'uos'>

The **Object inspector** shows the **module @ 0x3f40e45c** with the following attributes:

Name	Value
VfsFat	<class 'VfsFat'>
VfsLfs2	<class 'VfsLfs2'>
chdir	<function>
dupterm	<function>
dupterm_notify	<function>
getcwd	<function>
ilistdir	<function>
listdir	<function>
mkdir	<function>
mount	<function>
remove	<function>
rename	<function>
rmdir	<function>
stat	<function>
statvfs	<function>
umount	<function>
uname	<function>
urandom	<function>

The **Shell** window at the bottom shows the execution of the script:

```
>>> import uos
>>> uos.listdir()
['boot.py']
>>>
>>>
```

VYTVORENIE SCRIPTU

The screenshot shows the Thonny IDE interface. The main editor displays a Python script named `boot.py` with the following code:

```
1 import uos
2
3 print(uos.listdir())
```

A dialog box titled "Where to save to?" is open in the center, with two options: "This computer" and "MicroPython device". The "MicroPython device" option is highlighted with a red rectangle.

The left sidebar shows the file explorer with "This computer" and "MicroPython device" sections. The "MicroPython device" section contains a file named `boot.py`.

The right sidebar shows the "Variables" and "Object inspector" panels. The "Variables" panel lists the following variables:

Name	Value
-	['boot.py']
bdev	<Partition type=1, subtype=12>
gc	<module 'gc'>
uos	<module 'uos'>

The "Object inspector" panel shows the following objects:

Name	Value
VfsFat	<class 'VfsFat'>
Vfs.fat2	<class 'Vfs.fat2'>
chdir	<function>
dupterm	<function>
dupterm_notify	<function>
getcwd	<function>
ilistdir	<function>
listdir	<function>
mkdir	<function>
mount	<function>
remove	<function>
rename	<function>
rmdir	<function>
stat	<function>
statvfs	<function>
umount	<function>
uname	<function>
urandom	<function>

The bottom panel shows the shell output:

```
>>> import uos
>>> uos.listdir()
['boot.py']
>>>
>>>
```


OOP – MICROPYTHON (PYTHON)

atribút triedy

atribút objektu

```
[led.py]: x
1 from machine import Pin
2
3 class Led:
4     led_pin_default = 2
5
6     def __init__(self, pin_num = led_pin_default):
7         self.pin = Pin(pin_num, Pin.OUT)
8         self.pin.value(0)
9
10    def set(self, value):
11        self.pin.value(1) if value==1 else self.pin.value(0)
12        #if(value):
13            # self.pin.value(1)
14        #else:
15            # self.pin.value(0)
16
17    @classmethod
18    def print_default_pin(cls):
19        print(f"Default Led pin is : {cls.led_pin_default}")
20
21    @staticmethod
22    def sum_numbers(num1 , num2):
23        return num1 + num2
24
```

konštruktor

metóda triedy

statická metóda

```
Shell x
>>> from led import Led
>>> print(Led.led_pin_default)
2
>>> Led.print_default_pin()
Default Led pin is : 2
>>> Led.sum_numbers(1,2)
3
>>> ld = Led()
>>> ld.set(1)
```

ÚLOHA 1.

- Nahrajte najnovší micropython firmware na zariadenie ESP32.
- Vytvorte script, ktorý bude v 500 ms intervaloch blikať modrou LED (prípadne postupne rozsvetovať a zhasínať LED - PWM).
 - Použite triedu: `machine.Pin` + knižnicu `time` (prípadne `machine.PWM`).
- Vytvorte triedu `Led`, ktorá bude implementovať vyššie popísanú funkcionality.

ÚLOHA 2.

- Vytvorte script, ktorý zobrazí dostupné WiFi siete (SSID) + RSSI. Ak je sieť nezabezpečená, zobrazí pri danej sieti „open“. Následne vyzve používateľa, aby zadal SSID + heslo siete, ku ktorej sa chce pripojiť. Po pripojení zobrazí IP adresu, masku siete, default gateway.
 - Použite triedu: `network.WLAN`.
- Vytvorte triedu WiFi, ktorá bude implementovať vyššie popísanú funkčnosť.

Ukážka výstupu:

```
NETGEAR83          -77
Eduroam            -80
Wifri              -81      open
...
Enter the SSID:    NETGEAR83
Enter the PSWD:   123pswd321
Connected!
IP address:       192.168.1.23
mask:             255.255.255.0
Default gateway:  192.168.1.1
```

ÚLOHA 3.

- Vytvorte script, ktorý vytvorí AP s názvom siete : ESP32_Priezvisko. Zároveň zmení IP adresu a DG na: 192.168.1.1.
 - Vyskúšajte sa pripojiť mobilným telefónom.
 - Použite triedu: `network.WLAN`.
- Implementujte do triedy WiFi novú metódu, ktorá bude vykonávať vyššie popísanú funkcionality.

ĎAKUJEM ZA POZORNOST



Vytvorené v rámci projektu KEGA 026TUKE-4/2021