

Testovanie SW v Pythone

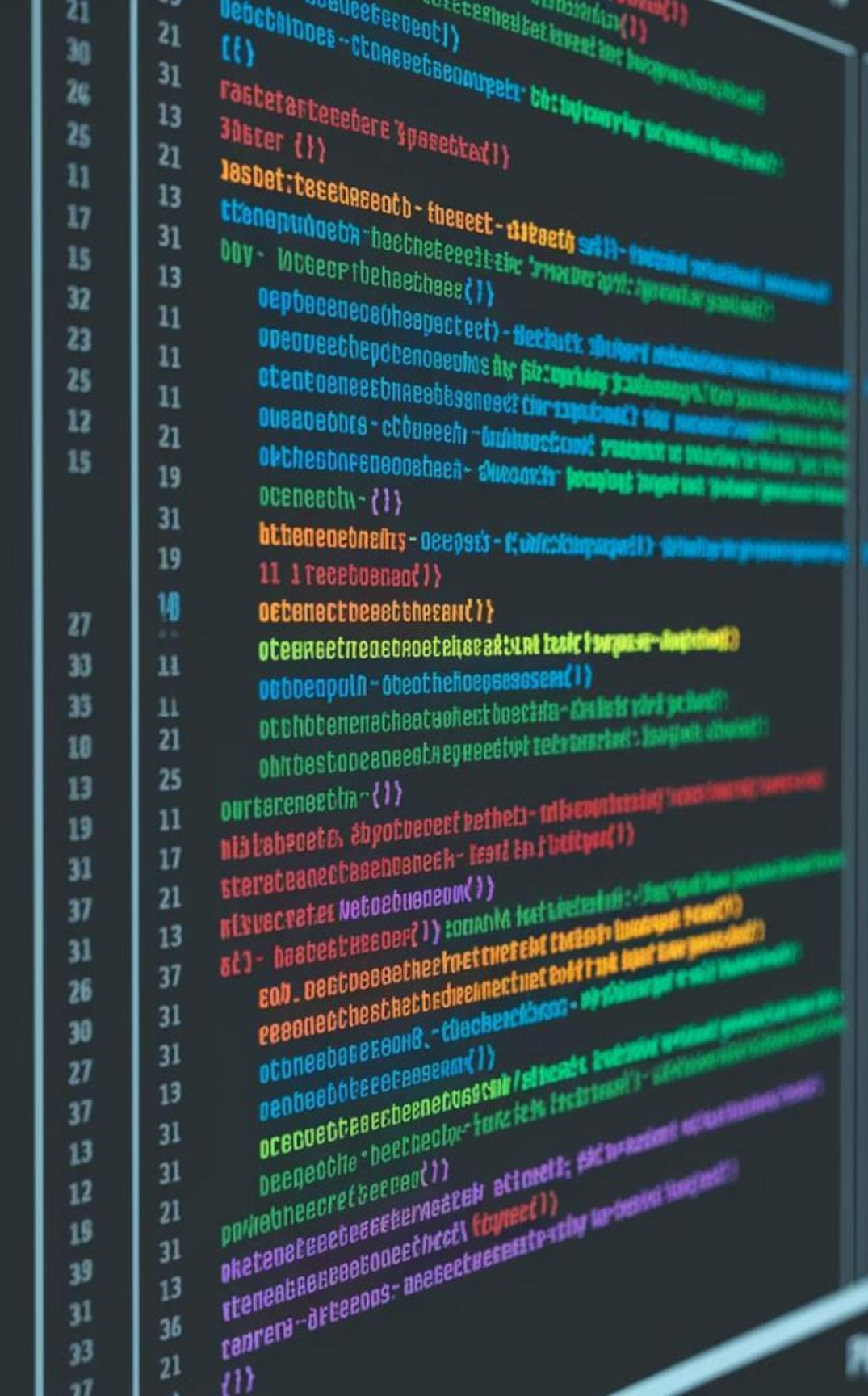
I. Začiatočník



Testovanie softvéru v Pythone I. Začiatočník

Vitajte v kurze zameranom na testovanie softvéru v jazyku Python. Naučíte sa základy testovania, prácu s unittest a pytest, a ako efektívne automatizovať testovanie vašich aplikácií.

 **Miroslav Reiter**



Ako Začneme?

1. Pridajte si ma na **LinkedIn**

www.linkedin.com/in/miroslav-reiter

2. Stiahnite si Cvičný NTB Súbor

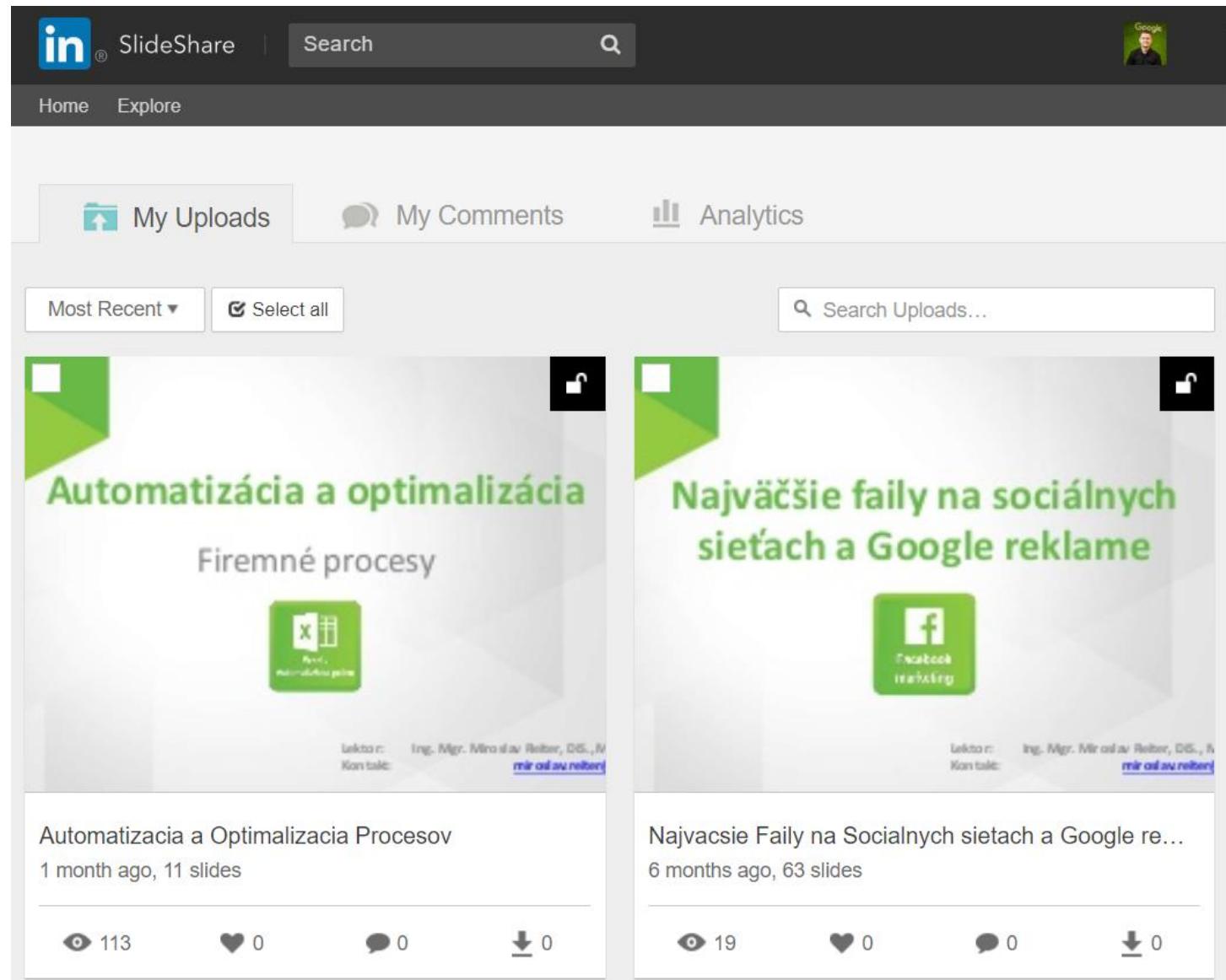
github.com/miroslav-reiter/Kurzy_SAV_Analytika_Python_R

3. Prezentácia po prednáške

<https://github.com/miroslav-reiter/>

4. Videá na YouTube

www.youtube.com/@VITA-Academy





miroslav-reiter

 Type / to search

Overview

Repositories 108

Projects

Packages

Stars 115

Sponsoring



Miroslav Reiter

miroslav-reiter

Founder of VITA Academy ★ Microsoft Certified Trainer ★ Google, Android Certified Trainer ★ ISTQB Trainer

[Edit profile](#)

78 followers · 2 following

VITA Academy

Bratislava

miroslav.reiter@it-academy.sk

www.vita.sk

<https://orcid.org/0000-0003-1804-651X>

@VITA_Academy_SK

in/miroslav-reiter

VitaAcademySK

@VITA-Academy

miroslav-reiter / README.md



About me

- 🌟 Hi, I am @Miroslav-Reiter
- 🎓 I am a Google Certified Trainer GCT, Microsoft Certified Trainer MCT, Microsoft Most Valuable Professional MVP, ISTQB/GASQB Trainer, PRINCE2/ITIL4/ArchiMate/TGAF/UML/BPMN/Scrum Trainer
- 💻 My preferred programming languages are Java 🍒, Python 🐍, JavaScript (Google Apps Script), VBA and R
- 🎓 I teach programming and how to use IT technologies effectively at [VITA](#) and [IT Academy](#) My online courses can be bought directly at [VITA.sk](#). I recommend buying an annual subscription with all courses
[Online Akreditované Kurzy a Skolenia VITA](#)
- 🗣 I frequently speak at conferences and workshops. Mostly about IT (programming, automation, certification, testing, enterprise architecture and modeling), management (Project management, PRINCE2, ITIL4), education and online marketing with ads (Google Ads, Google Analytics, Copywriting, Social media, YouTube)
- 🎥 I regularly upload a new videos to [YouTube channel Miroslav Reiter - VITA Academy](#)
 9.4k
- 💡 Fun fact: I have graduated from 14 different Universities. I am currently studying at 4 other Universities, 3 of which are PhDs.
Check out my [LinkedIn](#) profile

My Socials

 [YOUTUBE](#) [VITA](#) [FACEBOOK](#) [LINKEDIN](#) [VITA ACADEMY](#) [LINKEDIN](#) [TWITTER](#) [INSTAGRAM](#)[Google](#) [VITA](#) [Google](#) [IT Academy](#)

Languages, Tools and Frameworks





miroslav-reiter / Testovanie_Softveru

Type / to search

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#) Testovanie_Softveru Public[Pin](#)[Unwatch 1](#)[Fork 1](#)[Starred 1](#)[main](#)

1 Branch 0 Tags

[Go to file](#)[Add file](#)[Code](#)[About](#) miroslav-reiter Add files via uploadacc131c · 5 minutes ago [36 Commits](#) Cvicenia

Add files via upload

2 months ago

 Poziadavky_Tester

Add files via upload

3 months ago

 Zoznam_SW

Delete Zoznam_SW/a

3 months ago

 ! Testovanie Softvéru v Python Zdrojové Kódy.z...

Add files via upload

5 minutes ago

 README.md

Update README.md

2 months ago

 README

Testovanie Softvérū

Materiály k online kurzu Testovanie Softvérū

Požiadavky na SW Testerov, QA Manažérov a Test Analytikov

1. <https://www.alianciasr.sk/najziadanejsie-nedostatkove-pracovne-pozicie-na-trhu-prace/>
2. <https://www.sustavapovolani.sk/register-zamestnani/pracovna-oblast/karta-zamestnania/17936-ikt-tester/>
3. <https://www.nsp.cz/jednotka-prace/softwarovy-tester>
4. <https://www.nsp.cz/jednotka-prace/tester-automatizovaneho-t>
5. https://www.surveymonkey.com/r/SOQR25?utm_source=website&utm_medium=login-page
6. <https://www.howtodeal.dev/>
7. <https://neilonsoftware.com/>

How to Deal with Difficult People on Software Projects

Product Managers



The Dictator



The Sales Liaison



The Executive Assistant



The Napkin Sketcher



The Scope Wiggle



The Patent Author



The Scope Creeper



The People Pleaser

Materiály k online kurzu Testovanie Softvérū

[www.vita.sk/](#)[testing](#) [testing-tools](#) [software-testing](#)
[testing-tool](#)[Readme](#)[Activity](#)[1 star](#)[1 watching](#)[1 fork](#)

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

Languages

 Gherkin 100.0%



Kurzy SAV DataScience, Python, R, Julia, BI, AI/ML, ChatGPT

Materiály, Zdrojové Kódy a Projekty, Prezentácie ku kurzom SAV DataScience, Python, OOP, R, BI, Analytika

Python je interpretovaný, interaktívny, open-source programovací jazyk. Python beží na mnohých variantoch Unixu, na Macu a Windowse (súčasťou kurzu bude inštalácia na vašom systéme). Pre absolvovanie kurzu je potrebné mať k dispozícii vlastný notebook (s ľubovoľným operačným systémom podporujúcim Python).

R je programovací jazyk a softvérové prostredie pre štatistickú analýzu a vizualizáciu. R je voľne dostupný pod GNU licenciou a existujú verzie pre mnohé operačné systémy ako Linux, Windows a Mac.

ChatGPT je pokročilý jazykový model umelej inteligencie vyvinutý spoločnosťou OpenAI. Je to variant modelu GPT (Generative Pre-trained Transformer), ktorá je špeciálne navrhnutá na generovanie textu a interakciu v dialógovej

About



Materiály, Zdrojové Kódy, Prezentácie ku kurzom SAV Python, OOP, R, BI, Data Science

python jupyter numpy oop
 jupyter-notebook pandas python3
 matplotlib sav beautifulsoup reiter
 matplotlib-pyplot

Readme

Activity

22 stars

16 watching

23 forks

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

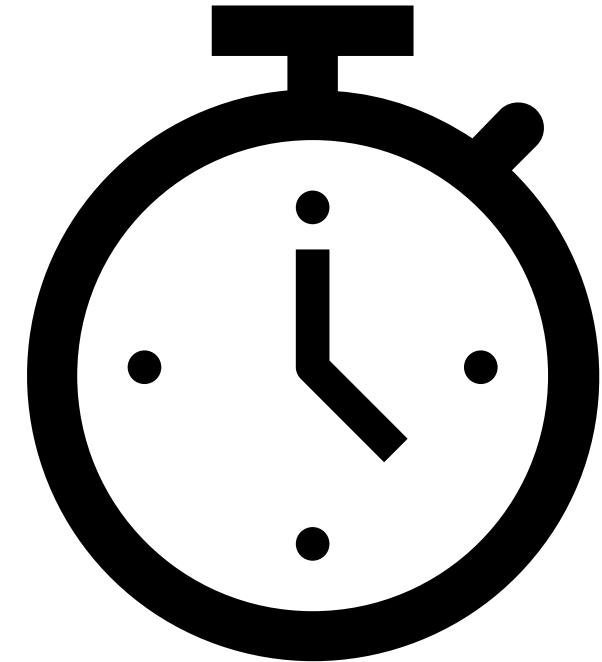
Languages



Suggested workflows

Úvodné Informácie

- Časový rozvrh (9:00-13:30)
 - Programujeme/Testujeme (50 min)
 - Prestávky (10 min)
 - Obedová prestávka
 - Mobilné telefóny a zariadenia
-
- Priprav si otázky a rovno sa pýtaj
 - Interaktívna forma



O mne - Miroslav Reiter

8

**40000+
klientov a
1000+ firiem**

**IT Architekt
Programátor
Manažér**

**Microsoft
Google
ISTQB tréner**

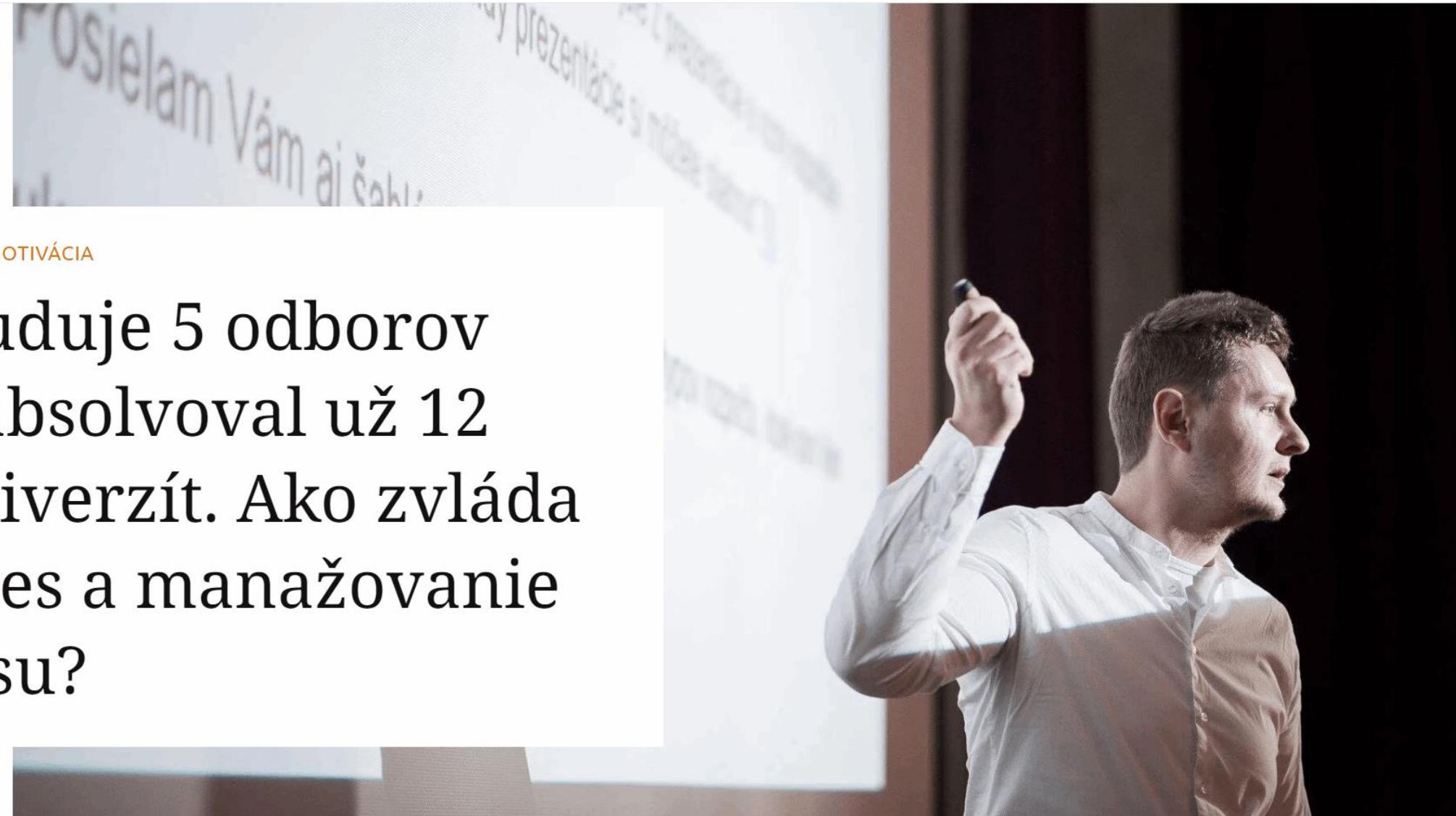
134 certifikácií

**151 príručiek a
publikácií**

14 škôl

62 projektov

Vlastná firma



MOTIVÁCIA

Študuje 5 odborov a absolvoval už 12 univerzít. Ako zvláda stres a manažovanie času?

Foto: Jakub Kovalík pre FMK UCM | Miroslav Reiter na prednáške Grow with Google na FMK UCM.



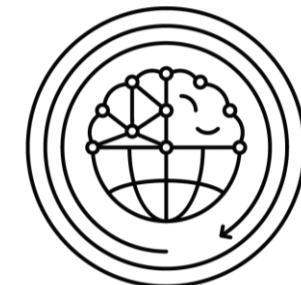
Nikola Kotláriková
19. júl 2022 · 8 min. čítania



Miroslav Reiter¹⁰



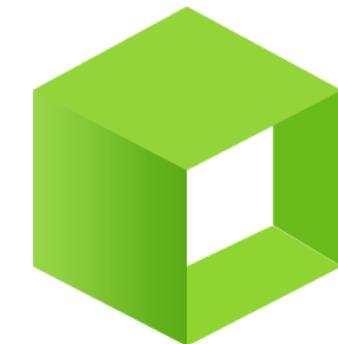
1. PhDr. VŠM (Podnikovný manažment)
2. Ing. STU FEI (**Aplikovaná informatika**)
3. Mgr. UK FM (**Strategický manažment a marketing**)
4. Mgr. VŠM (**Manažérstvo kvality**)
5. Mgr. VŠEMVŠ (Verejná správa)
6. Mgr. DTI (**Učiteľstvo ekonomických predmetov**)
7. DiS. AMOS (Cestovný ruch)
8. **MBA LIGS (Executive management)**
9. **DBA Humanum (IT manažment)**
10. MPA IES (Verejná správa a samospráva krajov)
11. MSc. Humanum (**Bezpečnosť inf. systémov**)
12. Ing. Paed. IGIP STU
13. Mgr. PEVŠ (**Bezpečnosť informačných systémov**)
14. RNDR. PEVŠ (**Bezpečnosť informačných systémov**)



FAKULTA MANAGEMENTU
Univerzita Komenského
v Bratislave

**DIGITÁLNA
UNIVERZITA**

STU
FIIT



IT ACADEMY

 **VITA**
ACADEMY

Most Valuable Professionals

[About](#) [Events](#) [Find an MVP](#)[Profile](#) [Events](#)

Headline

 IT Architect and Programmer  Hard worker 
Lecturer and Certified Trainer

Biography

 I'm a hard worker, intellectual and joker. I love learning and teaching as well. My main objective is to teach people and improve their IT knowledge. To create truly practical knowledge necessary for life and present it in an interesting way. I don't like snake charmers and people who cannot...

[▽ Read more](#)

Miroslav Reiter

 Slovakia IT Academy s.r.o.

Most Valuable Professionals

High Impact Activities

Award Category

M365

Technology Area

Visio, Excel

Languages

English, Slovak

Social



This community leader has not added a high impact activity yet.

 [Domov](#) > [Registre](#) > [Znalcí](#) > PhDr. Ing. Miroslav Reiter

Znalec

PhDr. Ing. Mgr. Miroslav Reiter

Evidenčné číslo: 915864

Miesto výkonu činnosti

Tomášikova 50
83104 Bratislava
Slovenská republika

[Zobraziť na mape](#)

Kontaktné údaje

Mobil: +421 908 163 084
E-mail: znapec@it-academy.sk

Odbory a odvetvia

| Odbor / Odvetvie | História | Stav |
|---|---------------------------|----------------|
| 100000 - Elektrotechnika | | |
| 100400 - Riadiaca technika, výpočtová technika (hardware) | 14.02.2023 - Zápis | AKTÍVNY |
| 100800 - Nosiče zvukových a zvukovoobrazových záznamov | 14.02.2023 - Zápis | AKTÍVNY |
| 100900 - Počítačové programy (software) | 14.02.2023 - Zápis | AKTÍVNY |
| 101000 - Bezpečnosť a ochrana informačných systémov | 14.02.2023 - Zápis | AKTÍVNY |

Vyberte si online kurz

Naučte sa programovať, tvoriť webstránky a grafiku, manažovať alebo sa zamerajte na osobný rozvoj. Všetko jednoducho vďaka našim online kurzom z pohodlia domova.

Ročné Predplatné na všetky Online Kurzy

2290 €

490 €

Prístup pre Vás do všetkých Aktuálnych aj
Pripravovaných Online Kurzov

12 mesačná platnosť

Kúpiť teraz

Zistiť viac



560 kurzov v ponuke



Zábavné online lekcie



Akreditované kurzy



13 rokov skúseností



Certifikovaní profesionálni lektori

Odporúčame Kurzy špeciálne pre vás



Online kurz SAP I.
Začiatočník
~~298,00€~~ 398,00€



Balík SAP Profesionál
~~998,80€~~ 998,80€



Online kurz ChatGPT a DALL-E AI
~~186,00€~~ 254,00€



Online kurz SQL I.
Začiatočník
~~196,00€~~ 276,00€



Program MPA Manažér Štátnej a Verejnej Správy
~~1 940,00€~~ 2 540,00€



Online kurz Lektor (Akreditovaný Kurz Lektor)
~~238,00€~~ 296,00€

Chat

Všetko Videá Obrázky Správy Krátke videá Knihy Web :: Viac

Nástroje

[www.youtube.com › watch](http://www.youtube.com/watch)

Ako začať programovať v Pythone? - Online kurz Python a ...



Akreditovaný online kurz Python I. Začiatočník: →
<https://www.vita.sk/online-kurz-python-i-zaciatocnik/> Akreditovaný prezenč...

YouTube · Miroslav Reiter - VITA Academy · 10. 11. 2021

16 kľúčových momentov v tomto videu

www.youtube.com/watch

Online Kurz Python I. Začiatočník - Ukážka - IDE, Pycharm ...

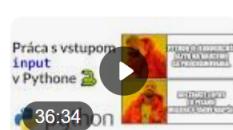


Online Kurz Python I. Začiatočník - Ukážka - IDE, Pycharm, Print, Premenné, Konverzie, Netbeans · Comments6.

YouTube · Miroslav Reiter - VITA Academy · 2. 3. 2020

10 kľúčových momentov v tomto videu

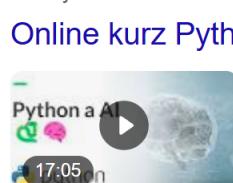
[www.youtube.com/watch](http://www.youtube.com/watch?v=...)



Prednášajúci: PhDr. Ing. Mgr. et Mgr. et Mgr. et Mgr. Miroslav Reiter, DiS., MBA, MPA, MSC, DBA, Ing., Paed., IIGIP. Miesto: SAV v Bratislave.

YouTube : Miresley Reiter - VITA Academy - 25.8.2023

4 klúčové momenty v tomto video



... **Online kurz Python** - Používanie AI pri Programovaní v Pythone a Generovanie Zdrojových Kódov Akreditovaný online kurz Programovanie

YouTube - Mirrored Reiter - VITA Academy - 25.9.2022

[www.youtube.com/watch](http://www.youtube.com/watch?v=...) Online kurz Python - Filozofie, Princípy a Dokumentácia



Comments4 ; Online kurz Python - Cyklus While, break a continue. Miroslav Reiter - VITA Academy ; 156 views ; Online kurz Jazyk Julia - Ako

[Všetko](#) [Videá](#) [Obrázky](#) [Správy](#) [Krátke videá](#) [Web](#) [Knihy](#) [Viac](#)[Nástroje](#)

m.youtube.com › watch

Online Kurz Testovanie Softvéru I. Začiatočník - Požiadavky ...



Akreditovaný online kurz Testovanie Softvéru I. Začiatočník: →
<https://www.vita.sk/online-kurz-testovanie-softveru-i-zaciatochnik/> ...

YouTube · Miroslav Reiter - VITA Academy · 4. 3. 2020

13 klúčových momentov v tomto videu ▾

www.youtube.com › watch

Online kurz Testovanie Softvéru podľa ISTQB pre ...



Akreditovaný online kurz Testovanie Softvéru podľa ISTQB →
<https://www.vita.sk/online-kurz-testovanie-softveru-podla-istqb-i-...>

YouTube · Miroslav Reiter - VITA Academy · 6. 9. 2023

3 klúčové momenty v tomto videu ▾

www.youtube.com › watch

Online kurz Ako sa stať Testerom



Akreditovaný online kurz Ako sa stať Testerom: → <https://www.vita.sk/online-kurz-testovanie-softveru-i-zaciatochnik/> Akreditovaný prezenčný ...

YouTube · Miroslav Reiter - VITA Academy · 3. 9. 2021

16 klúčových momentov v tomto videu ▾

www.youtube.com › watch

Online kurz Testovanie Softvéru podľa ISTQB - Ad Hoc Testy ...

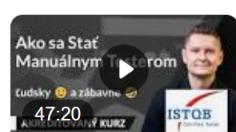


Online kurz Testovanie Softvéru podľa ISTQB I. Začiatočník je ideálny pre tých, ktorí sa chcú naučiť základy testovania softvéru.

YouTube · Miroslav Reiter - VITA Academy · 22. 11. 2023

www.youtube.com › watch

Online kurz Manuálne Testovanie Softvéru - Ako sa stať ...



Akreditovaný online kurz Manuálne Testovanie Softvéru: →
<https://www.vita.sk/online-kurz-testovanie-softveru-i-zaciatochnik/> Online...

YouTube · Miroslav Reiter - VITA Academy · 11. 7. 2023



Luigi, Mário
a Yoshi

Čo Robíte?

1. Študent/Učiteľ

2. Zamestnanec

3. Podnikateľ

4. Nezamestnaný/materská

5. Dievča pre všetko



Vaše Ciele a Očakávania

1. Doplniť si znalosti z jazyka Python

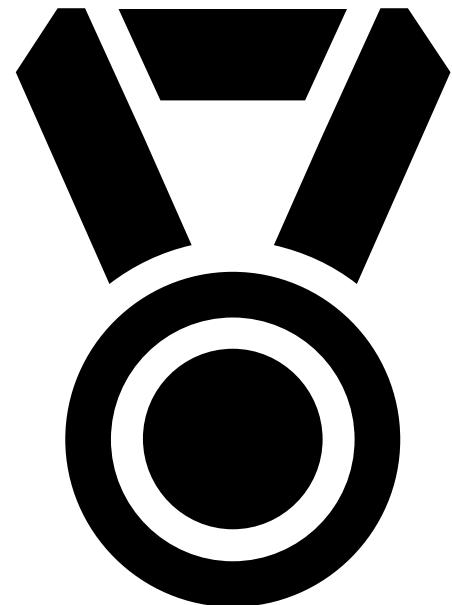
2. Naučiť sa testovanie softvéru

3. Práca jednotkovými testami (unittest)

4. Práca s Robot Framework

5. Mockovanie

Zábava je v zaručená v každom bode 😊



Ako začať so SW Testami v Pythone 2

AKREDITOVANÝ KURZ





Čo sa naučíme?

1. Čo je testovanie softvéru?
2. Aké máme typy testovanie?
3. Rozdiely (Manuál vs. Automat)?
4. Ako sa zorientovať v testovacích nástrojoch v Python?
5. Čo si treba nainštalovať?
6. Aké sú klúčové moduly?
7. Kde nájdeme dokumentáciu?
8. Profit



Čo je testovanie softvéru?



Overenie funkčnosti

Testovanie overuje, či softvér funguje správne podľa špecifikácií.



Odhalenie chýb

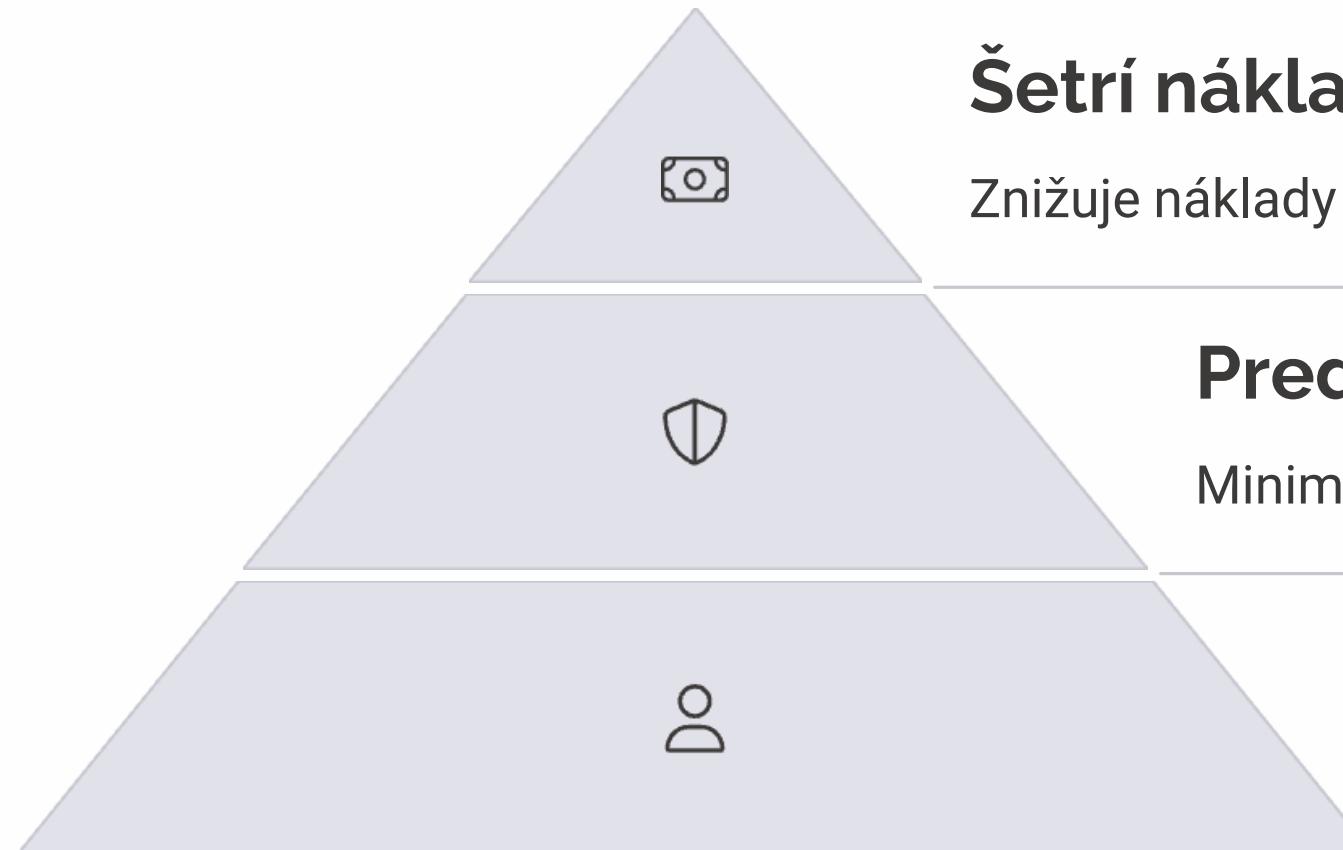
Pomáha odhaliť chyby ešte pred nasadením do produkcie.



Zvýšenie kvality

Buduje dôveru v kvalitu aplikácie a minimalizuje náklady na opravy.

Prečo je testovanie dôležité?



Šetrí náklady

Znižuje náklady na opravy vo finálnej fáze

Predchádza incidentom

Minimalizuje problémy v produkcií

Zvyšuje spokojnosť

Zlepšuje kvalitu produktu

DPH - датів під час тестирування

```
dphi - ouuns, spizard -  
1  
1 (tersseek - ont - ()  
    Sostteer(ooinofisopere:ibue/laase:(ožiney)  
    postesesse)))  
    prcesses)))  
1 )  
1  
# TEST CASE"  
    (touseet^ to hed nies nse faðg hrælee)  
    (tfoeshrreo 8 fialfeesr avane ðapazdy filtnes)  
        Sbastceeoine:nofare:idee:iaotprersprey[bainerid:laase:(ðaispnee]  
        seattessses)))  
1 1 1 )  
1 (tersseek - tuðH oontes moet - hour - ()  
    seattret(oollspare:(oe"lsporce:(oleispreey)  
    epattesser)))  
    Sastreaseee:iinapærri:oprtee:laſ:(ðalne·case)  
1 1 1 )  
1  
1 (tersseek - dat - ()  
    Sastreoftree:(oateett.Ioose:(opere:(oefialspee)y)  
    Sstatey))  
    prctesttor(oer(oollsparey))  
1 )  
1 (tersseek - tuðH oontir maise haisy to·read - ()  
    Shstteei(oelispfee-laise:liaipere:(ožire:(ðalne-ðaispnee)y)  
    psattesser)))  
    prcesses)))  
1 )  
1  
# TEST CASE"  
    sosS - dat - ()  
    Raascer(ooolooifiopere:ibue/laase:(ožine:(oaispnee)y)  
    srcesssesse)))  
    prcesses)))  
    srceesttey(oaree,(opore:(ožiney))  
1 )
```

Príklad testovania v praxi

Funkcia

vypocitaj_dph(100)

Očakávaný výsledok

120 (pri 20% DPH)

Prínos testu

Zabezpečí, že aj po zmene kódu sa správanie nezmení.

Typy testovania softvéru

Manuálne testovanie



- Vykonáva tester ručne
- Vhodné pre malé testy
- Ideálne pre UX testy
- Náročné na čas

Automatizované testovanie

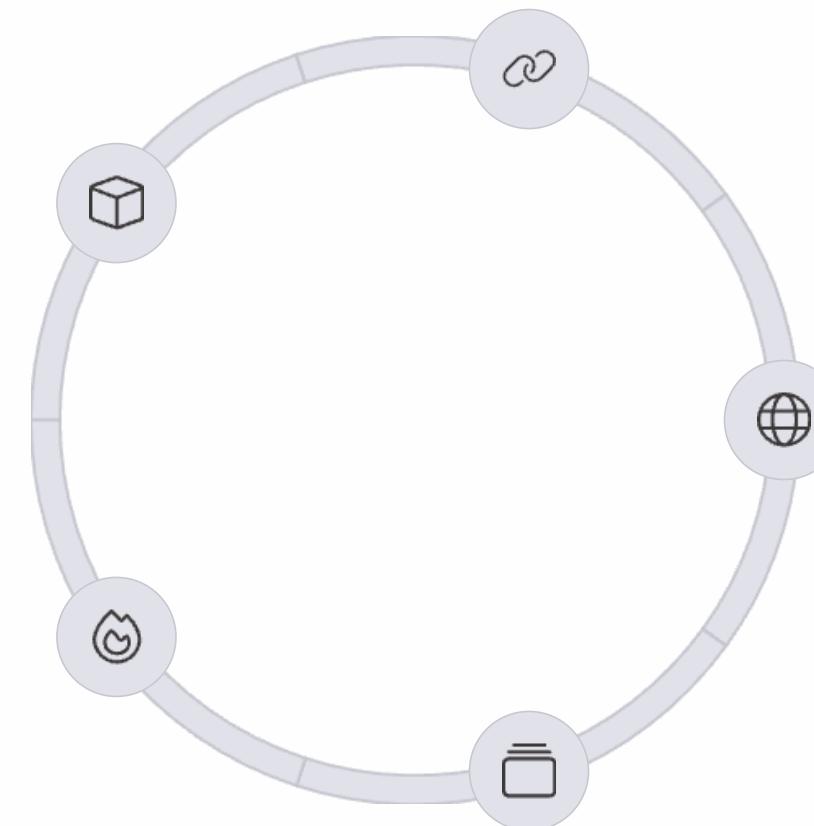


- Pomocou skriptov a nástrojov
- Rýchle a opakovateľné
- Ideálne pre regresné testy
- Vyššia počiatočná investícia

Typy testov v praxi

Unit testy
Testovanie jednotlivých funkcií a metód

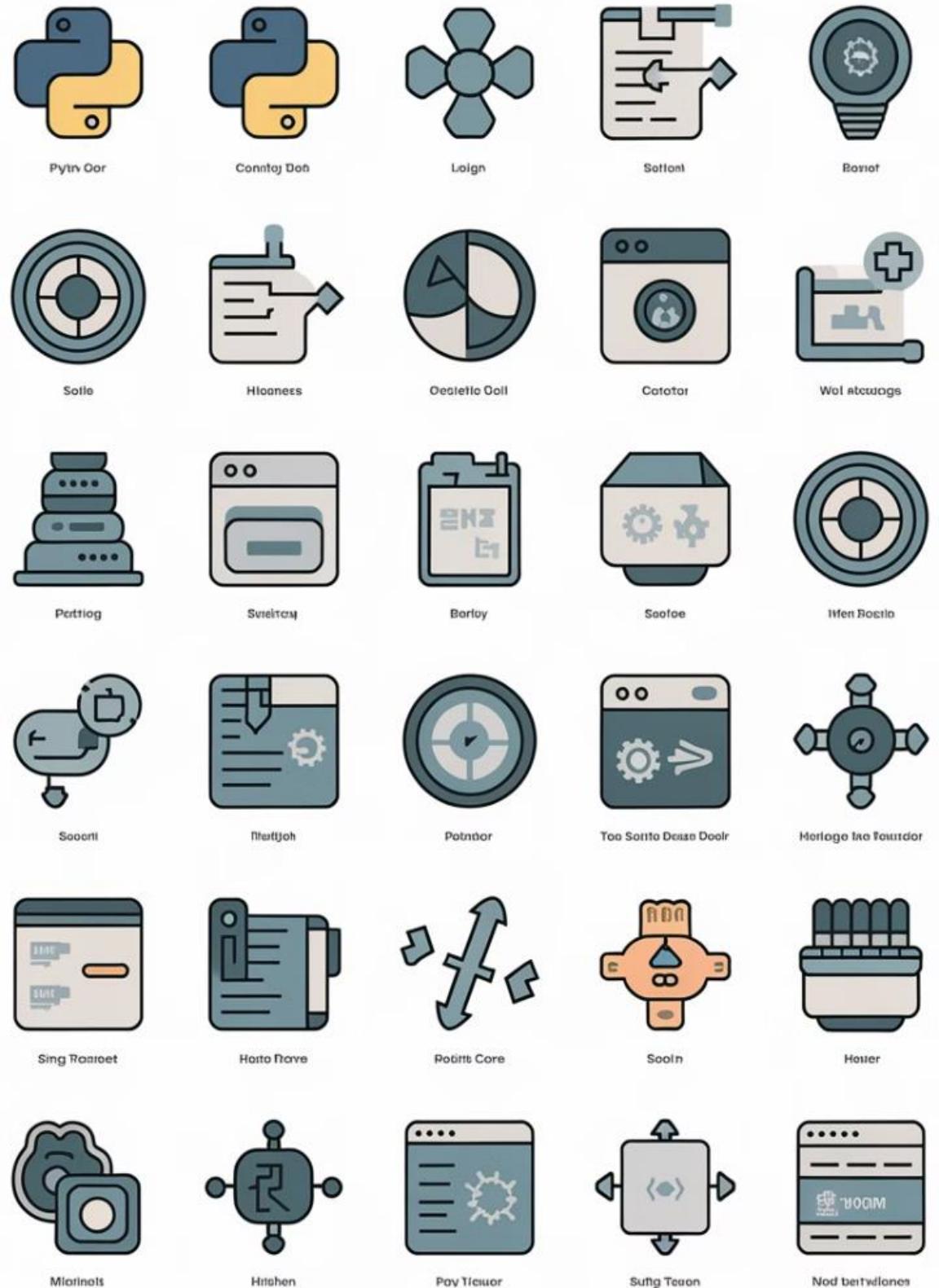
Smoke testy
Rýchle základné testy po nasadení



Integration testy
Testovanie prepojenia modulov

End-to-End testy
Testovanie aplikácie ako celku

Regression testy
Overenie, že nová úprava nič nepokazila



Nástroje na testovanie v Pythonie

| Nástroj | Typ | Využitie | Výhody |
|-----------------|-----------|---------------------|---------------------------------------|
| unittest | vstavaný | unit testy | stabilita, súčasť Pythonu |
| pytest | framework | unit + integrácia | jednoduchý zápis, fixture, pluginy |
| Selenium | web | E2E testovanie webu | klikanie, input, navigácia |
| Robot Framework | všeobecný | keyword-driven | ľahko čitateľný, aj pre netechnických |

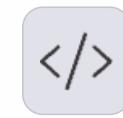
```
Ipython ( unittest)
lunte) +ttoypottt+ mple ce
lpotest t+o+I→I+copcempee+)
ipytest tcd)
lunte) +tfotpeitbest)
ipythosthomn+1+ mcekene
lpnte) ((ecwauetteempef+)
ipythont+o
lpnte) +ttoyo)
lpvtest ttotpeitbest)
lpote) +toot+I→I+coueempet+)
lpntocit+o
```

unittest – základný testovací modul



Súčasť štandardnej knižnice

Nulová závislosť od externých knižníc.



Inšpirovany JUnit

Testy sa definujú v triedach ako metódy.



Bohatá funkcia

K dispozícii setUp(), tearDown(), assert* metódy.

Príklad unittest kódu

```
import unittest

class TestVypocet(unittest.TestCase):
    def test_dph(self):
        self.assertEqual(vypocitaj_dph(100), 120)
```

o o o
..
H
Z
M
C
H
I
R
H
T
S
ate-te)
< teste-nittate.
dt:))
test.xlas-my-function
unittest testcase)
at:
< . . :fnn-tt)
4 tes-my-function-
my-function(2)-
returns-4)
ket.

pytest – moderný a výkonný framework

Jednoduchý zápis

Nevyžaduje triedy – stačia funkcie.

Pokročilé funkcie

Podpora fixture, parametrize, assert rewriting.

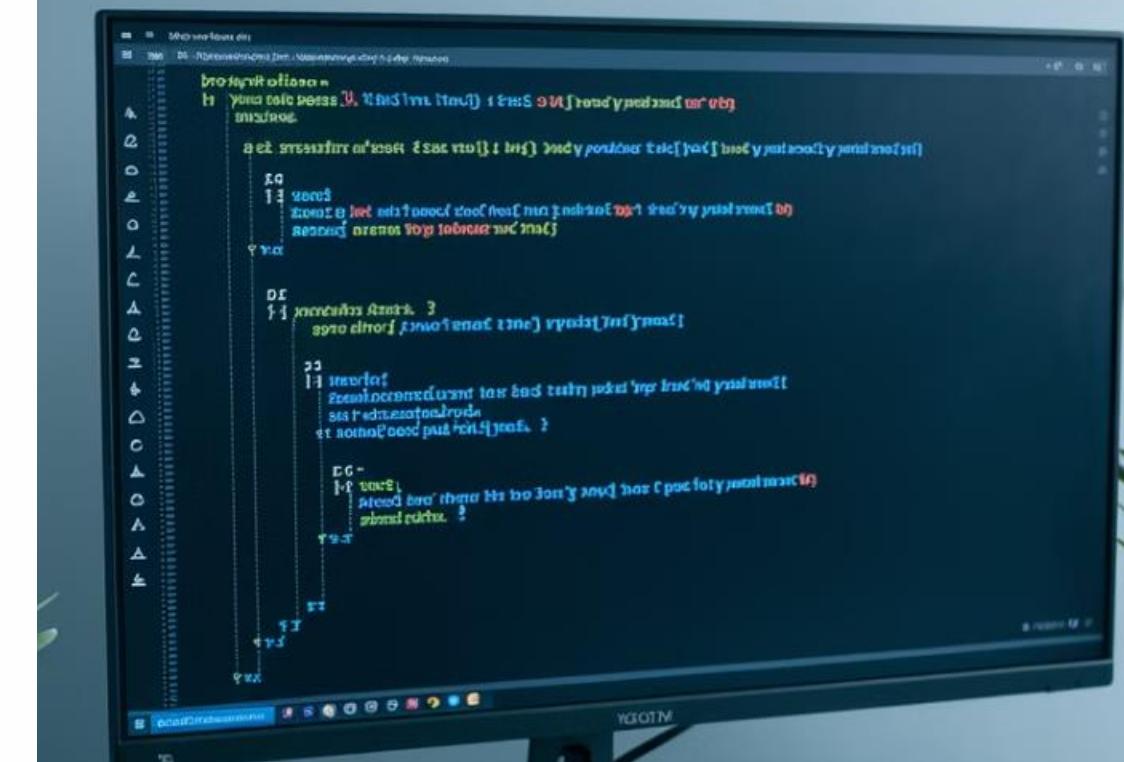
Ideálny pre komplexné projekty

Skvelý pre väčšie a komplexné projekty.

Príklad pytest kódu

```
def test_dph():
    assert vypocitaj_dph(100) == 120
```

Všimnite si jednoduchosť zápisu v porovnaní s unittest. Nie je potrebné vytvárať triedy ani používať self.assertEqual.





Selenium – automatizácia webových testov



Ovláda reálny prehliadač

Pracuje s Chrome, Firefox a ďalšími cez WebDriver.

Testuje UI

Umožňuje kliknutia, zadávanie údajov, validácie.

Webové aplikácie

Ideálne pre testovanie webových aplikácií.



Selenium IDE

[Pridať do Chrome](#)[seleniumhq.org](#) 3,5 ★ (288 hodnotení) [Zdieľať](#)

Rozšírenie

Nástroje pre vývojárov

600 000 používateľia

i Toto rozšírenie už nie je k dispozícii, pretože nie je v súlade s osvedčenými postupmi pre rozšírenia pre Chrome.

[Ďalšie informácie](#)

Vyskúšajte jednu z týchto alternatív:



TestCase Studio - Selenium...
5,0 ★ (184) i

TestCase Studio record the user actions performed on a web...



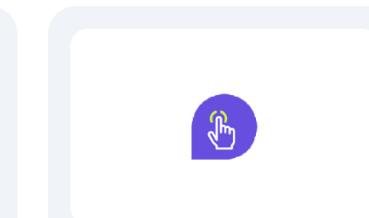
Katalon Recorder (Seleniu...
4,2 ★ (263) i

Selenium IDE alternative to record and export Selenium scripts. Wit...



Selenideium Element...
5,0 ★ (4) i

A handy tool to log attributes and Selenide, Selenium, Cypress,...



CSS Selector Helper
3,9 ★ (63) i

Dev Tools sidebar that aids finding unique CSS selectors for Seleniu...

The screenshot shows the Selenium IDE interface running in a browser window. On the left, there's a sidebar with a tree view of 'Selenium Projects' containing various test cases. The main area displays a recorded sequence of steps: '1. open /', '2. click at link=Projects', '3. click at link=Selenium WebDriver', '4. click at link=Projects', '5. click at link=Selenium IDE', and '6. click at link=Browser Automation'. Below this, there are sections for 'Selenium WebDriver' and 'Selenium IDE' with descriptions and links. A central part of the interface shows a preview of a web page with a green checkmark icon.

The screenshot shows the Selenium IDE interface running in a browser window. The layout is similar to the previous one, with a sidebar for projects and a main area for test cases. The test cases listed are: '1. open /', '2. click at link=Projects', '3. click at xpath=//*[contains(text(),'Selenium WebDriver')]', '4. click at link=Projects', '5. click at link=Selenium IDE', '6. click at link=Projects', and '7. click at xpath=//*[contains(text(),'Selenium Grid')][2]'. The interface includes sections for 'Selenium WebDriver' and 'Selenium IDE' with their respective descriptions and links.



Katalon Recorder (Selenium tests generator)

[Pridať do Chromu](#)

Vybrané 4,2 ★ (263 hodnotení) Zdieľať

[Rozšírenie](#)[Prac. postup a plánovanie](#)

100 000 používateľia

The screenshot shows the Katalon Recorder extension interface. It features a toolbar with buttons for New, Record, Play, Play Suite, Play All, Pause, and Export. Below the toolbar is a table titled 'Test Suites' with one entry: 'all*' under 'demo'. The table columns are 'Command', 'Target', and 'Value'. The 'Command' column lists actions like 'type', 'click', 'select', and 'submit'. The 'Target' column lists element IDs such as 'id-company', 'id-role', and 'label-Nice manager/Lead'. The 'Value' column lists corresponding values like 'company', 'label-Technical Architect', and 'label-Excellent colleague'. At the bottom of the interface, there are tabs for Log, Screenshots, Variable, Test Data, and Reference, along with a code editor window displaying recorded Selenium-like script logs.

Katalon Recorder Tutorial

Automation Testing with Katalon Recorder

Katalon Recorder

Powerful recorder to capture & edit automation scripts.

Manage and Run test suite easily.

Get automation tools at katalon.com

Automation Testing with Katalon Recorder

Prehľad

Selenium IDE alternative to record and export Selenium scripts. With reports & screenshots. Fast & open-source.

Katalon Recorder is a free, lightweight web extension that serves as an alternative to Selenium IDE. Designed for automating browser actions and tests, Katalon Recorder is available on Chrome.

With Katalon Recorder, you can:

Ensure quality by testing your work before passing it on to the QA team.

Maintain end-to-end software quality through automated testing.

Monitor web application functionality using synthetic testing.

Track testing activities with shareable reports, complete with visual dashboards and charts.

Save time by automating repetitive browser tasks such as generating reports, filling out forms, and even automating web-based games.



Príklad Selenium kódu

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("https://example.com")
assert "Example" in driver.title
```

Robot Framework – keyword-driven testovanie



Pre netechnických členov

Vhodný pre tímy s netechnickými členmi.



Čitateľný ako dokumentácia

Syntax: Keyword Argument1 Argument2



Rozšíriteľný

Má rozšírenia pre web, databázy, API a ďalšie.

```
Definice funkce pro kontrolu titulky:
Definice funkce pro kontrolu obsahu stránky:
Definice funkce pro kontrolu URL stránky:
Definice funkce pro kontrolu hodnoty v input políku:
Definice funkce pro kontrolu hodnoty v checkbox políku:
Definice funkce pro kontrolu hodnoty v dropdown menu:
Definice funkce pro kontrolu hodnoty v textarea políku:
Definice funkce pro kontrolu hodnoty v radio button políku:
Definice funkce pro kontrolu hodnoty v select políku:
Definice funkce pro kontrolu hodnoty v file input políku:
Definice funkce pro kontrolu hodnoty v password input políku:
Definice funkce pro kontrolu hodnoty v date input políku:
Definice funkce pro kontrolu hodnoty v time input políku:
Definice funkce pro kontrolu hodnoty v week input políku:
Definice funkce pro kontrolu hodnoty v month input políku:
Definice funkce pro kontrolu hodnoty v number input políku:
Definice funkce pro kontrolu hodnoty v range input políku:
Definice funkce pro kontrolu hodnoty v checkbox políku:
Definice funkce pro kontrolu hodnoty v dropdown menu:
Definice funkce pro kontrolu hodnoty v radio button políku:
Definice funkce pro kontrolu hodnoty v select políku:
Definice funkce pro kontrolu hodnoty v file input políku:
Definice funkce pro kontrolu hodnoty v password input políku:
Definice funkce pro kontrolu hodnoty v date input políku:
Definice funkce pro kontrolu hodnoty v time input políku:
Definice funkce pro kontrolu hodnoty v week input políku:
Definice funkce pro kontrolu hodnoty v month input políku:
Definice funkce pro kontrolu hodnoty v number input políku:
Definice funkce pro kontrolu hodnoty v range input políku:
```

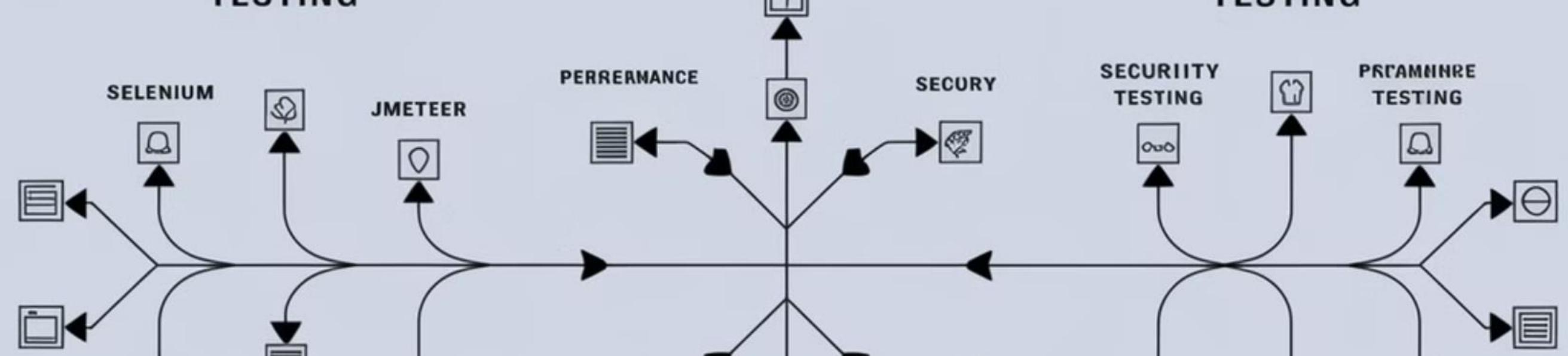
Príklad Robot Framework kódu

*** Test Cases ***

Kontrola titulky

Open Browser <https://example.com> Chrome

Title Should Be Example Domain



Kedy použiť ktorý nástroj?

 **unittest**
Pri malých/starších projektoch a v školskom prostredí.

 **pytest**
Moderný nástroj pre väčšinu Python aplikácií.

 **Selenium**
Pre testovanie webových stránok a používateľského rozhrania.

 **Robot Framework**
Ak sa do testovania zapájajú netechnické osoby.

Odporúčania do praxe

1 Začnite s pytest

Ideálny pre začiatočníkov v Pythone.

2 Kombinujte prístupy

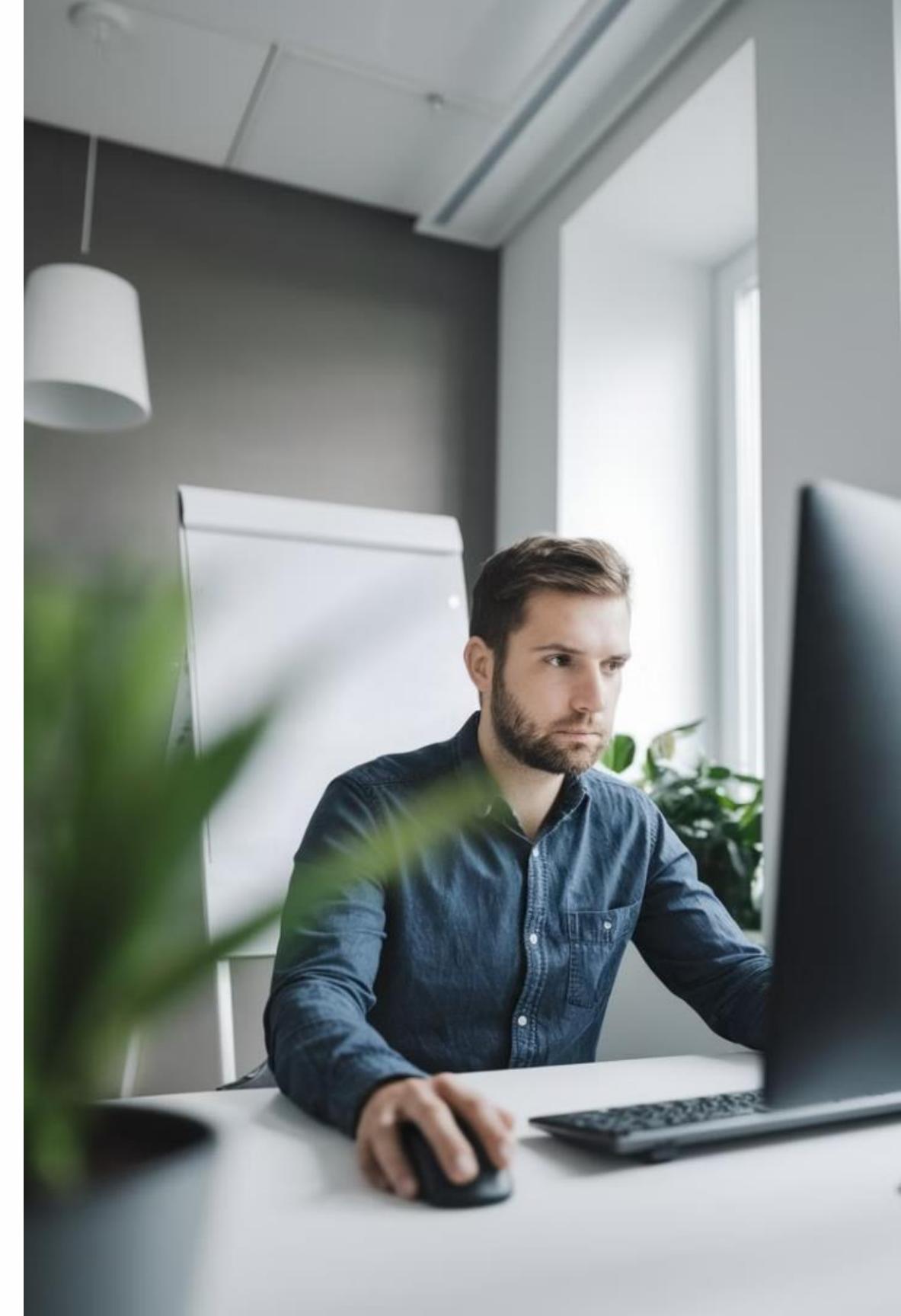
Manuálne aj automatizované testy majú svoje miesto.

3 Testujte často

Automatizujte a spúšťajte testy pri každom commite.

4 Pokrývajte okrajové prípady

Testujte aj chyby a výnimky, nielen štandardné scenáre.



Úlohy Testovanie sw Python

1. Čo je testovanie softvéru?
2. Aké máme typy testovanie?
3. Rozdiely (Manuál vs. Automat)?
4. Ako sa zorientovať v testovacích nástrojoch v Python?
5. Čo si treba nainštalovať?
6. Aké sú kľúčové moduly?
7. Kde nájdeme dokumentáciu?



Ako začať s Unit Testami v Pythone

AKREDITOVANÝ KURZ





Čo sa naučíme?

1. Čo sú to Unit Testy?
2. Čo je to modul unittest?
3. Ako sa používajú unit testy?
4. Ako vyzerá základná štruktúra?
5. Čo sú to assert metódy?
6. Aké sú vhodné prípady použitia?
7. Kde nájdeme dokumentáciu?
8. Profit

Základy Python unittest

Súčasť štandardnej knižnice

Dostupný od verzie Python 2.1 bez dodatočnej inštalácie.

Inšpirovany JUnit

Patrí do rodiny xUnit testovacích frameworkov.

Objektovo-orientovaný

Testy sú organizované ako triedy a metódy.

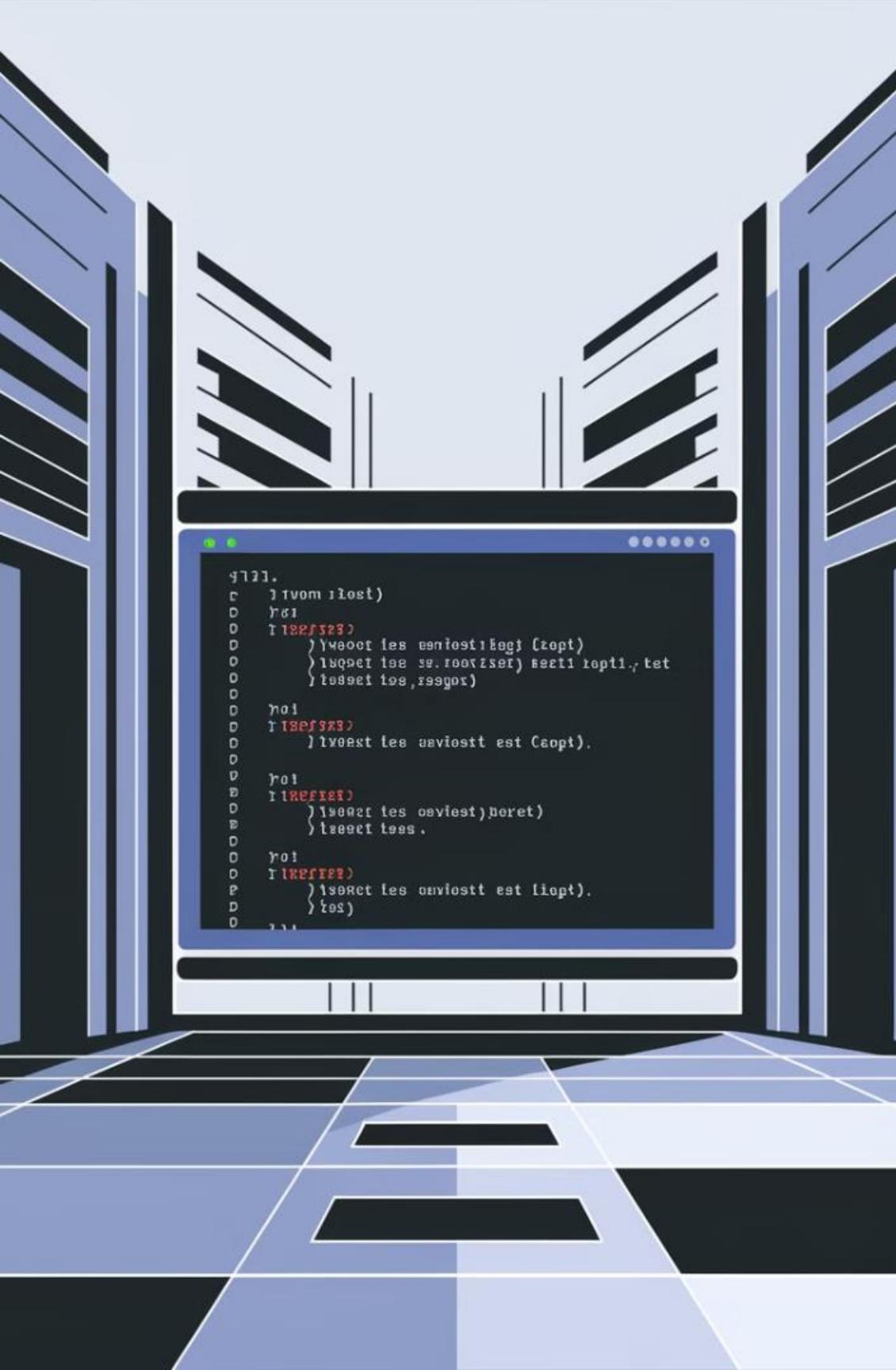


Vytváranie jednoduchých testovacích prípadov

```
import unittest

def sucet(a, b):
    return a + b

class TestMatematika(unittest.TestCase):
    def test_sucet(self):
        self.assertEqual(sucet(2, 3), 5)
```

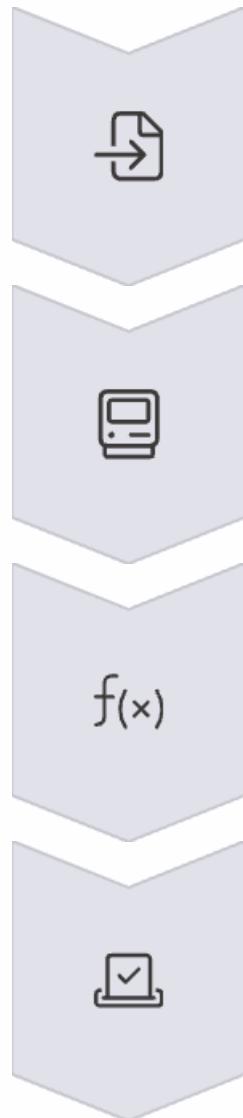


```
#!/usr/bin/python
# coding: utf-8
# This is a simple test of the addition function.
# It uses the built-in unittest module.

def sucet(a, b):
    """Vráti sumu dvoch čísel."""
    return a + b

class TestMatematika(unittest.TestCase):
    """Testy matematických funkcií."""
    def test_sucet(self):
        """Test súčtu dvoch čísel."""
        self.assertEqual(sucet(2, 3), 5)
```

Vysvetlenie základnej štruktúry



Import unittest

Načítanie testovacieho modulu.

Testovacia trieda

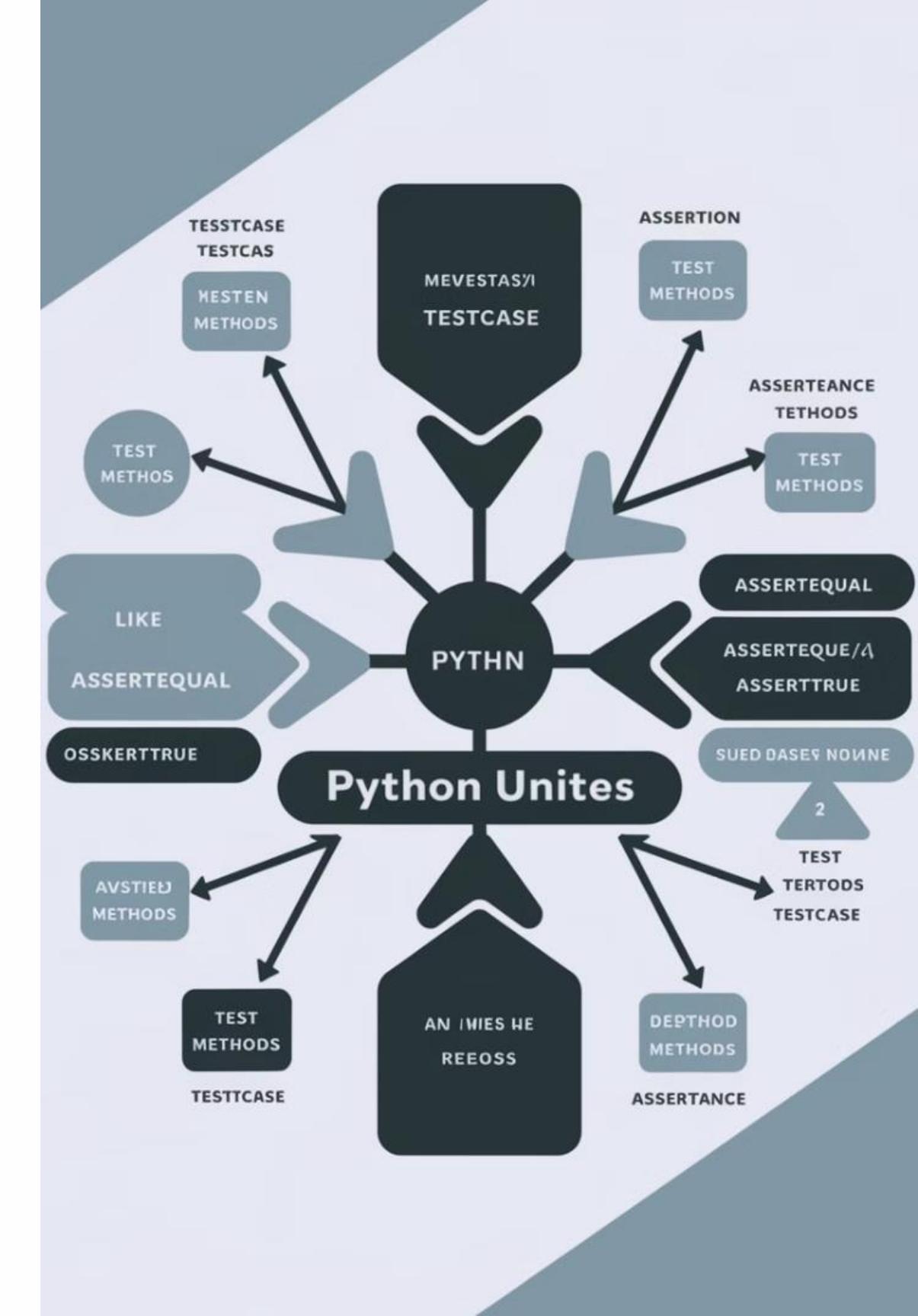
Dedí z unittest.TestCase.

Testovacia metóda

Začína prefixom test_.

Overenie výsledku

Pomocou self.assertEqual.



setUp() a tearDown()

setUp()

```
def setUp(self):  
    self.data = [1, 2, 3]
```

Vykoná sa pred každým testom.

tearDown()

```
def tearDown(self):  
    del self.data
```

Vykoná sa po každom teste.

Asserttual

AssertTrue

AssertFalse

AssertFalse

AssertFalse

Assertisqual

Assertfgreater

Assertisinstance

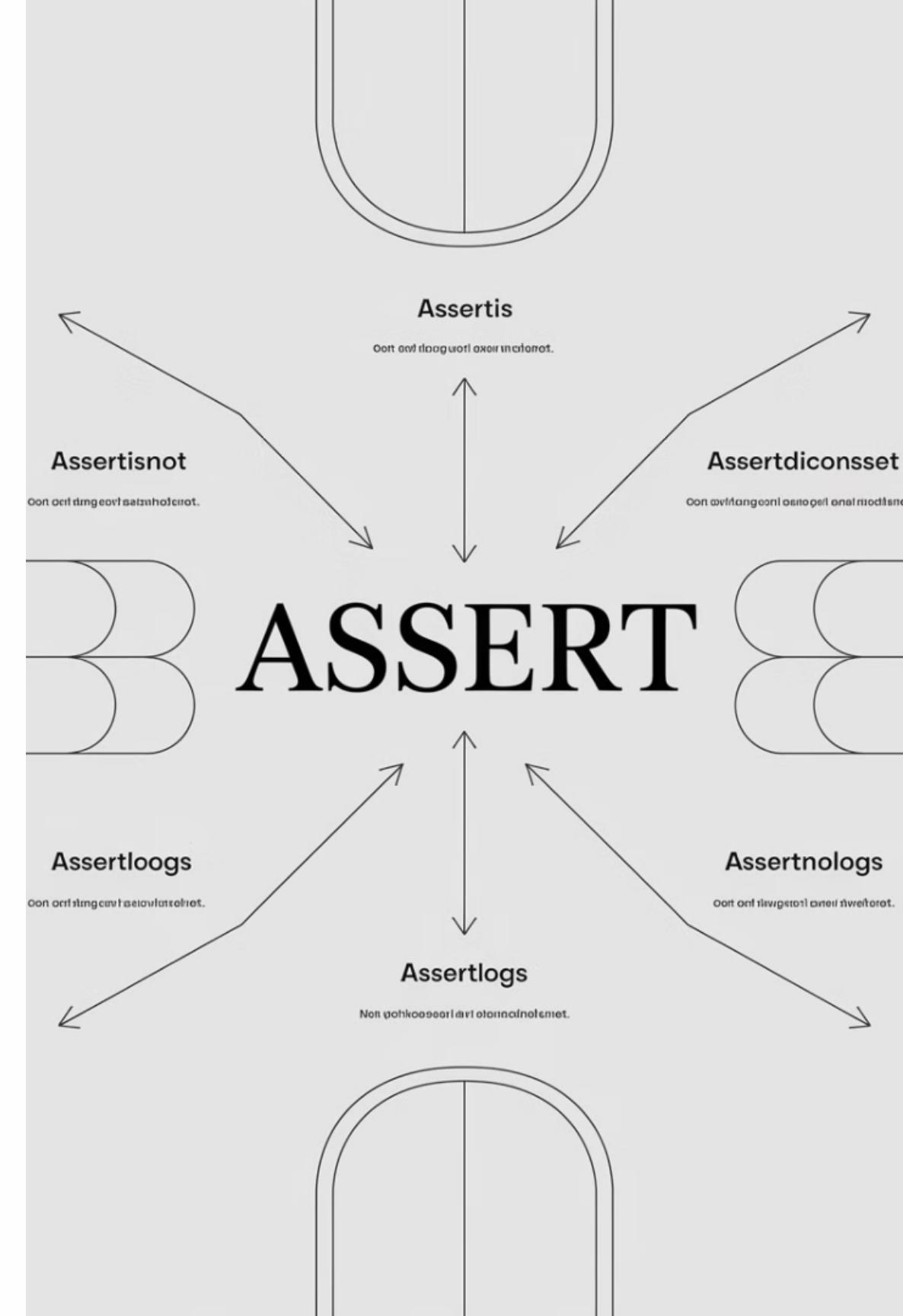
AssertisNone

Assert metódy v unittest

| Metóda | Použitie | Príklad |
|----------------------|------------------|----------------------|
| assertEqual(a, b) | a == b | assertEqual(5, 5) |
| assertNotEqual(a, b) | a != b | assertNotEqual(5, 4) |
| assertTrue(x) | bool(x) is True | assertTrue(1 < 2) |
| assertFalse(x) | bool(x) is False | assertFalse(3 < 1) |

Ďalšie assert metódy

| Metóda | Použitie | Príklad |
|-----------------|-------------------|---|
| assertIsNone(x) | x is None | assertIsNone(None) |
| assertIn(a, b) | a in b | assertIn(3, [1,2,3]) |
| assertRaises | očakávaná výnimka | with self.assertRaises(ValueError): int("abc") |



Spúšťanie testov

Cez terminál

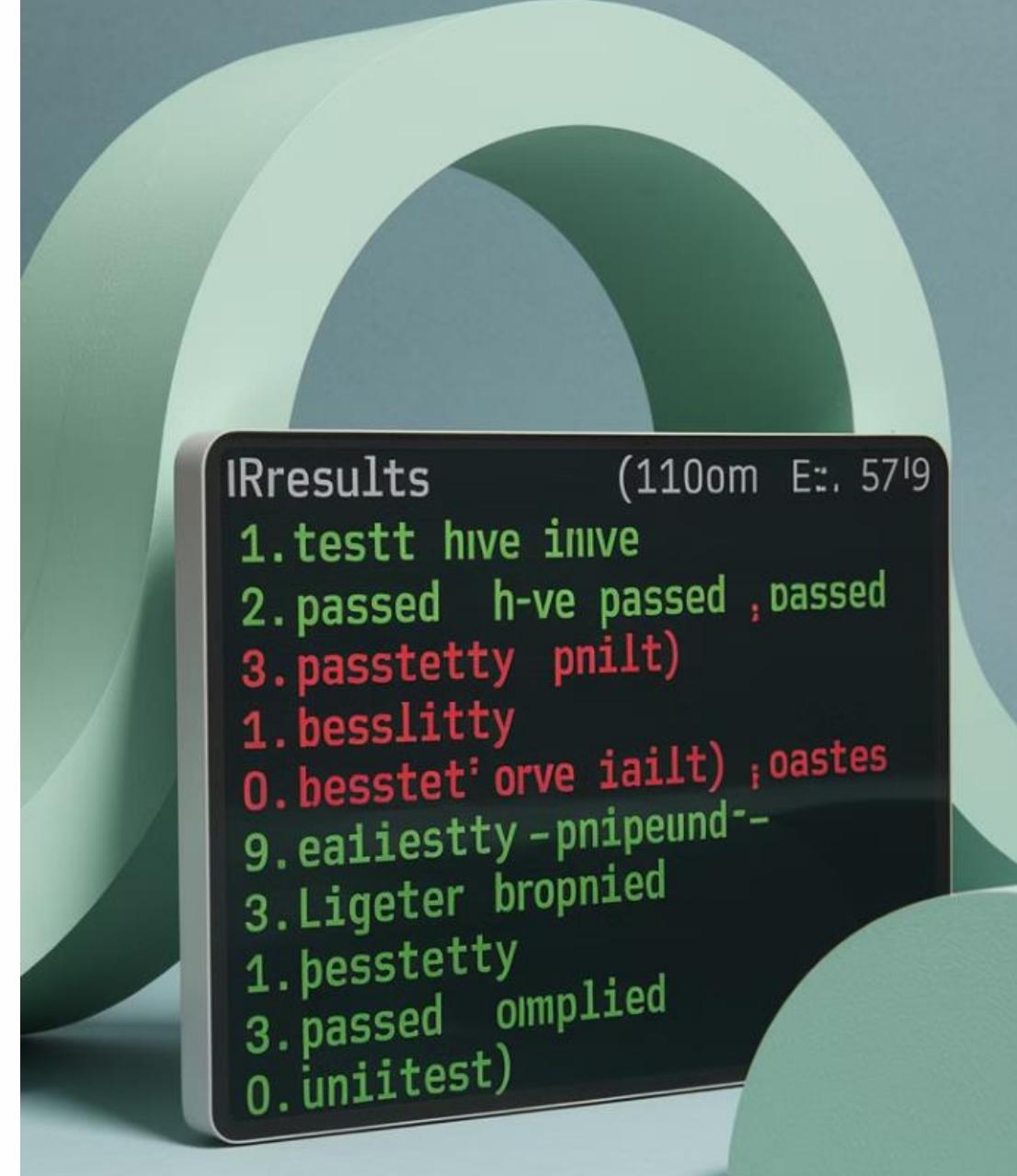
```
python -m unittest  
test_moj_modul.py
```

Priamo v skripte

```
if __name__ ==  
'__main__':  
    unittest.main()
```

Automatické vyhľadávanie

```
python -m unittest discover
```



```
1. testt hive inve  
2. passed h-ve passed ,passed  
3. passtetty pnilt)  
1. besslitty  
0. besstet' orve iailt) ;oastes  
9. eaiiestty -pnipeund--  
3. Ligeter bropnied  
1. besstetty  
3. passed oimplied  
0. uniitest)
```

Interpretácia výsledkov testov

Úspešný test

```
.
```

```
Ran 1 test in 0.001s
```

```
OK
```

Neúspešný test

```
F
```

```
FAIL: test_sucet
(test_moj_modul.TestMatematika)
```

```
Traceback (most recent call last):
...
AssertionError: 4 != 5
```

```
Ran 1 test in 0.001s
```

FAILED (failures=1)

Odporúčania pre unittest v praxi



Organizácia kódu

Vytvárať samostatný adresár tests/ pre testy.



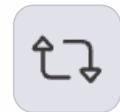
Pomenovanie

Pomenovať testy zrozumiteľne a výstižne.



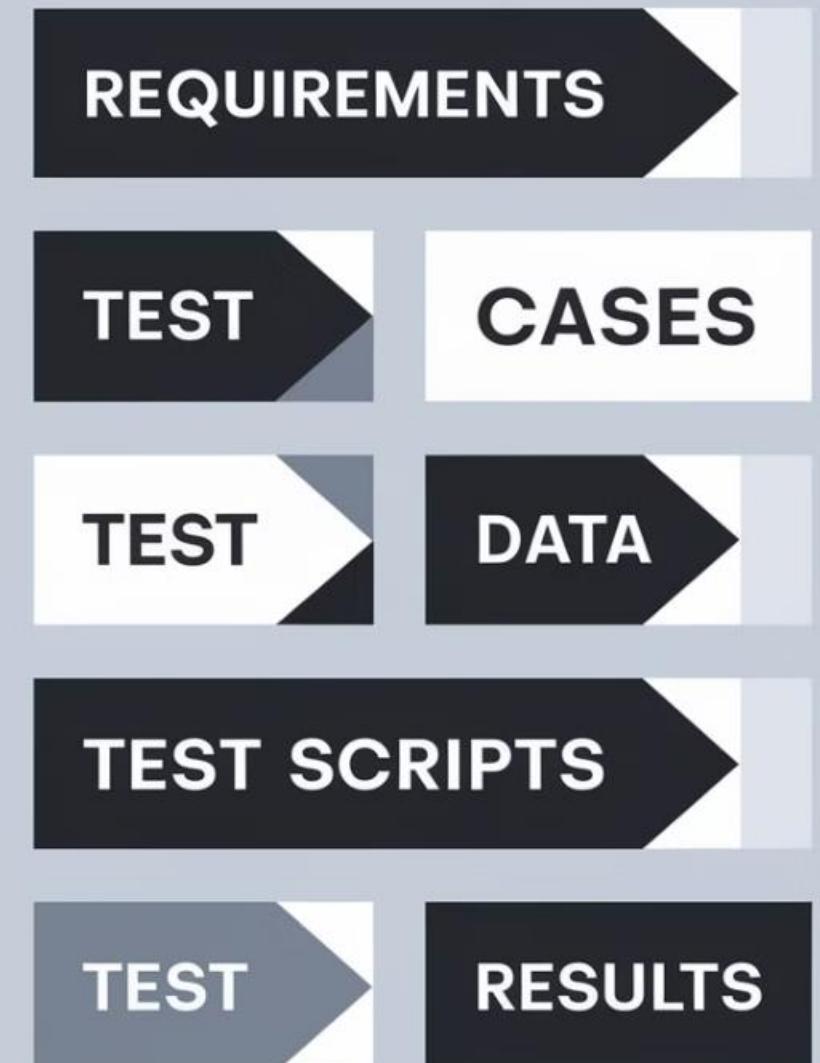
Modularita

Testovať každý modul a funkciu samostatne.



Častá exekúcia

Spúštať testy pri každej úprave kódu.



Štruktúra testovacieho súboru

```
import unittest

# testovaný kód
def rozdel_na_parne_neparne(zoznam):
    parne = [x for x in zoznam if x % 2 == 0]
    neparne = [x for x in zoznam if x % 2 != 0]
    return parne, neparne

# testy
class TestRozdelenie(unittest.TestCase):
    def test_porne_a_neporne(self):
        parne, neporne = rozdel_na_parne_neparne([1, 2, 3,
4])
        self.assertEqual(porne, [2, 4])
        self.assertEqual(neporne, [1, 3])

if __name__ == "__main__":
    unittest.main()
```

```
pettingerichtestivarekress(tatreeenbit)
teeritagnestirges&testtipestopat/tt>
zopfukersisatret #islien:
jeitekt/tt>
    11 ssteedisuatilicest tiecettufee-1>
    11     || sssz,?
    11     ssteetsrest/tt>
    22         if iostted/ llaaxited festt thone (bstryg: the: cspfue / t>
    22         corlouixazeflayaii:
    15             fxtier| testtrest testt lone sterx, festodlytirbm, 1:1>
    18                 jixes| worteast| testt/tt>
    16                     ||itvel|
    23                         ||{zzes:}
    11             neant lessst fosit[nestrestthane ftesg, festrreesttthue, 1:1>
    12                 ||feestthoh rnpstrionteese/tt>
    11             ribvctherriispet|
    18                 || sserdi tessr
    18                     ||at|oetry=1>
    10                     || ssteed festt loopffest stoxg, [het confrion t (bstrest / t>
    22                         |||
    11                         11
    23                         11
    22                         11
    14                         13
    18                         13
    18                         11
    25                         11
    11                         11
    18                         11
    11                         11
    23                         11
    13                         11
    12                         22
    22                         11
    25                         11
    15                         11
    11                         11
    11             ||sste.||festrest fest /het confrion : (bstrest / t>
    11                 fest teestfuestfesttthue festtrese /t>
    23                     ||tictz|
    11                     ||at|tugoste/tt>
    22                     ||fastees| worteast| testt lone sterx, festodlytirbm, 1:1>
    18                         ||acteed|
    25                         ||ssteel|
    18             neant lessst fn llerfustne /t>
    11             neastt festthob vogstatomihong ihace fosc{e /t>
    11 btest!>
    {98}>
```



Časté chyby v zápise



Chýbajúci self

Zabudnuté self ako prvý parameter metódy.



Nesprávny prefix

Testovacia metóda nezačína prefixom test_.



Nesprávny import

Import funkcie z nesprávneho modulu.



Nesprávny assert

Použitie obyčajného assert namiesto self.assertEqual.

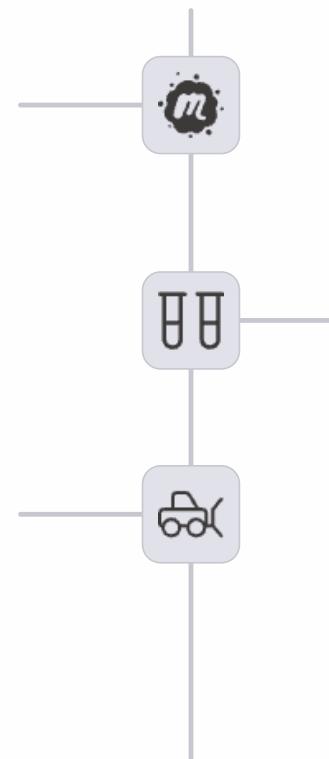
```
Ixs pØritete"lear"-  
shg he tepe"}-  
 <"ser>tpatat>)  
1?> assteque--  
 -<assertØxe"reas>  
 <teat"lØe  
 porttoperIØ .implemented  
 setØØ"nacktØ-}>  
 <ttup"1finaited  
 <test"Irure  
1?> nuensØ."""iØe---"
```



Životný cyklus testu

setUp()

Príprava testovacieho prostredia pred každým testom.

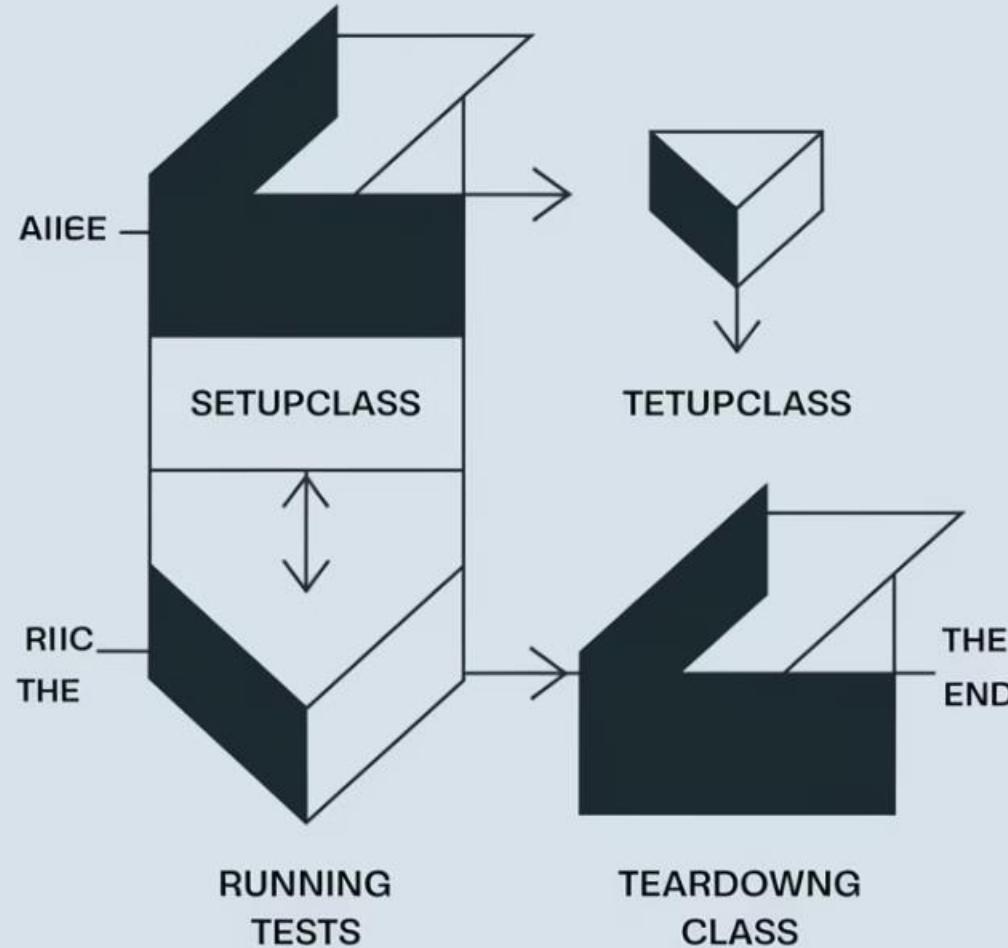


test_*()

Vykonanie samotného testu.

tearDown()

Upratovanie po každom teste.



Triedne metódy životného cyklu

| Metóda | Účel |
|------------------------------|---|
| <code>setUp()</code> | Príprava testovacieho prostredia pred každým testom |
| <code>tearDown()</code> | Upratovanie po každom teste |
| <code>setUpClass()</code> | Inicializácia na úrovni triedy (raz) |
| <code>tearDownClass()</code> | Upratovanie po všetkých testoch triedy |

Príklad setUpClass

```
@classmethod  
def setUpClass(cls):  
    print("Spúšťa sa raz na začiatku triedy")  
  
def setUp(self):  
    self.data = [1, 2, 3]  
  
def tearDown(self):  
    self.data.clear()
```

```
(@"LliWr"octodgupslast))  
sest v/idgecten"" Viic-  
((este-"wiidget"  
pr/LliWr"wwilddget""  
test :- ((octedgupclast))  
(@"LliWr"wiidget"  
pest v/idgectoddget"  
test v/idgectenc"(CTliast)
```

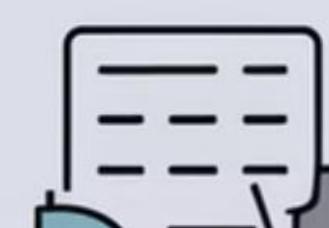
Assertequal()
(Isassertequal)

(assertisgequal)
(Isassertnotqual)

Assertistequal()
(Isassertnotocel)

Assertisnot()
(Isassertinl)

Assertisqual()
(Isassertnquel)



Základné assert metódy

assertEqual(a, b)

Overí, či a sa rovná b.

assertNotEqual(a, b)

Overí, či a sa nerovná b.

assertTrue(x)

Overí, či x je pravdivé.

assertFalse(x)

Overí, či x je nepravdivé.



Pokročilé assert metódy



assertIs(a, b)

Overí, či a je b (identita objektu).



assertIsNone(x)

Overí, či x je None.



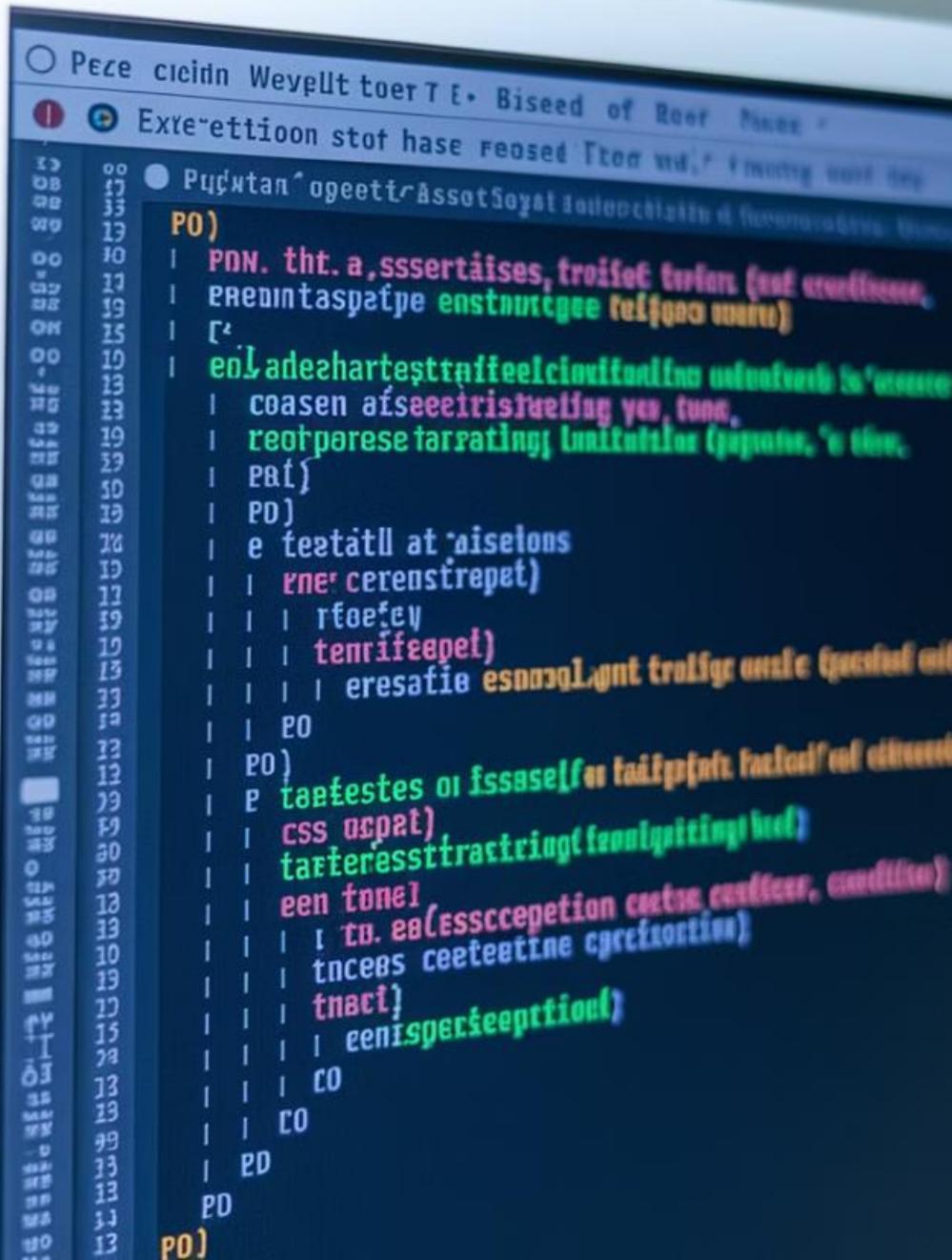
assertIn(x, iterable)

Overí, či x je v iterovateľnom objekte.

Očakávanie výnimiek

```
def test_vynimka(self):  
    with self.assertRaises(ZeroDivisionError):  
        vysledok = 1 / 0
```

Test je úspešný, ak kód vo with bloku vyhodí očakávanú výnimku ZeroDivisionError.



Prípadová parametrizácia

```
def test_vynasobenie(self):
    for a, b, vysledok in [(2, 3, 6), (0, 5, 0)]:
        with self.subTest(a=a, b=b):
            self.assertEqual(a * b, vysledok)
```

Unittest nemá priamo `@pytest.mark.parametrize`, ale možno použiť slučku alebo helper.

- “Testst
O pacmeterized,
P pcmnstoarau iotitthe t
O pon-aghe-(“EBSI1’,
P “Bovke clast,
Y “Test l pootrieraa třir,
T unctest yaus:
S (fliffece (“TY”)
A pcloodhe unlizes,
C Lonche (“tin”)
H pcœcn stperaze tread,
– – EprochoþT) I

Spúšťanie testov – prehľad možností

Príkazový riadok

```
python -m unittest test_modul.py  
python -m unittest discover -s tests -p  
"test_*.py"
```

IDE a integrácia

- VS Code: rozšírenie Python
- PyCharm: vstavaná podpora
- CI nástroje: GitHub Actions, GitLab CI



Výstup a interpretácia testov

Úspešný test

```
.
```

```
Ran 1 test in 0.002s
```

```
OK
```

Neúspešný test

```
F
```

```
=====
```

```
=====
```

```
FAIL: test_porne_a_neporne
(test_modul.TestRozdelenie)
```

```
=====
```

```
=====
```

```
Traceback (most recent call last):
  File "test_modul.py", line 9, in
    test_porne_a_neporne
    self.assertEqual(porne, [2, 5])
AssertionError: Lists differ: [2, 4] != [2, 5]
```

Odporúčania pre unittest v reálnych projektoch



Nezávislosť testov

Tvoríť testy samospustiteľné a nezávislé na poradí.



Konvencie pomenovania

Dodržiavať naming conventions: test_nazov_funkcie_pripad.



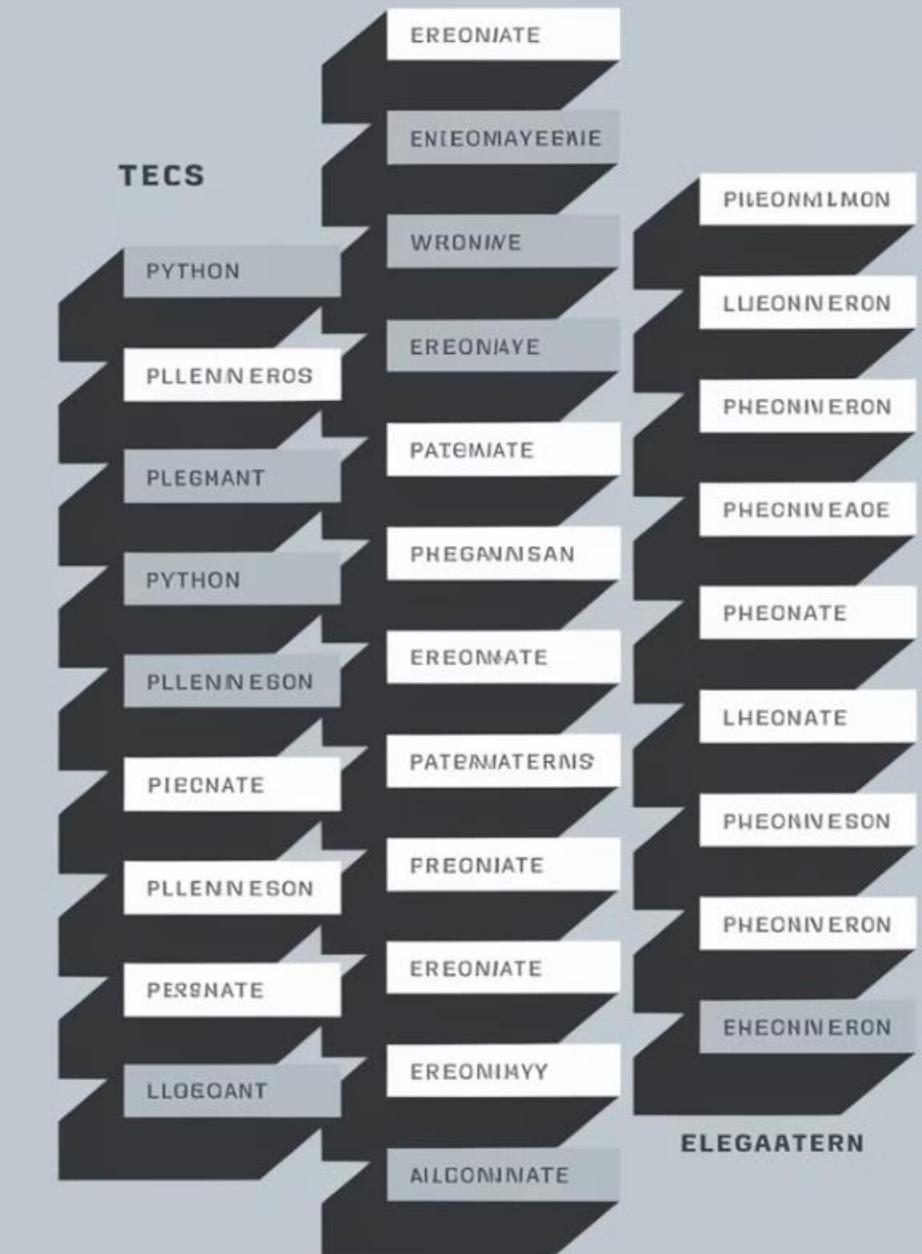
Kompletné pokrytie

Testovať pozitívne aj negatívne scenáre.



Organizácia kódu

Uchovávať testy v samostatnom adresári, napr. tests/.



Úlohy

Unit Testy

1. Čo sú to Unit Testy?
2. Čo je to modul unittest?
3. Ako sa používajú unit testy?
4. Ako vyzerá základná štruktúra?
5. Čo sú to assert metódy?
6. Aké sú vhodné prípady použitia?
7. Kde nájdeme dokumentáciu?



Ako začať s Python Robot Framework

AKREDITOVANÝ KURZ



ROBOT
FRAME
WORK /



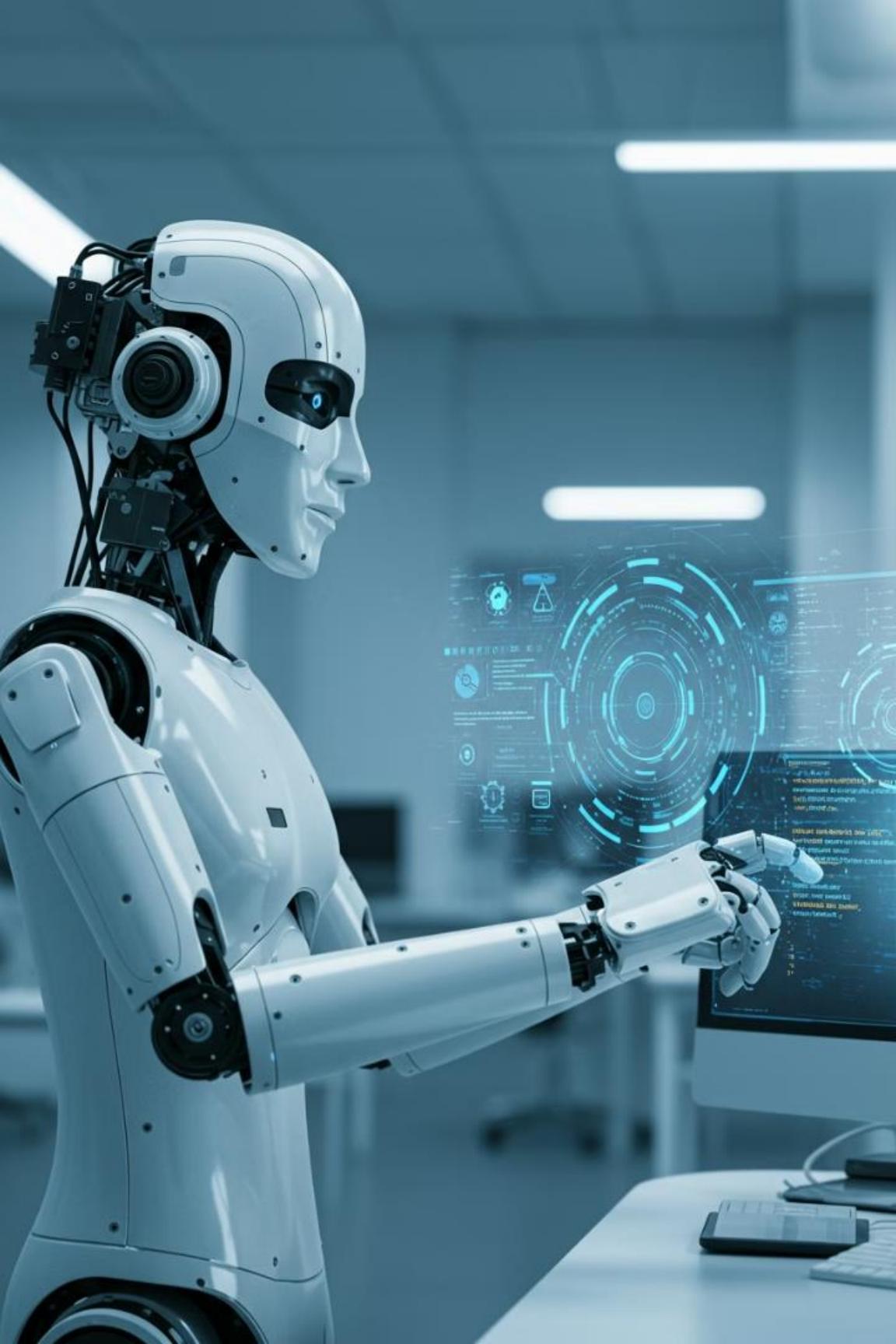
Robot Framework: Automatizované testovanie softvéru

Vitajte v kurze o Robot Framework. Tento open-source nástroj umožňuje efektívne automatizované testovanie softvéru.

Spoznajte jeho výhody, základnú štruktúru a praktické príklady použitia.



podle Miroslav Reiter





Čo sa naučíme?

1. Čo je to Robot Framework?
2. Ako sa nainštaluje a používa?
3. Ako sa v ňom tvoria testy?
4. Ako vyzerá základná štruktúra?
5. Čo sú to resources/variables?
6. Aké sú vhodné prípady použitia?
7. Kde nájdeme dokumentáciu?
8. Profit

Čo je Robot Framework?



Open-source framework

Vyvíjaný v jazyku Python, rozšíriteľný vlastnými knižnicami.



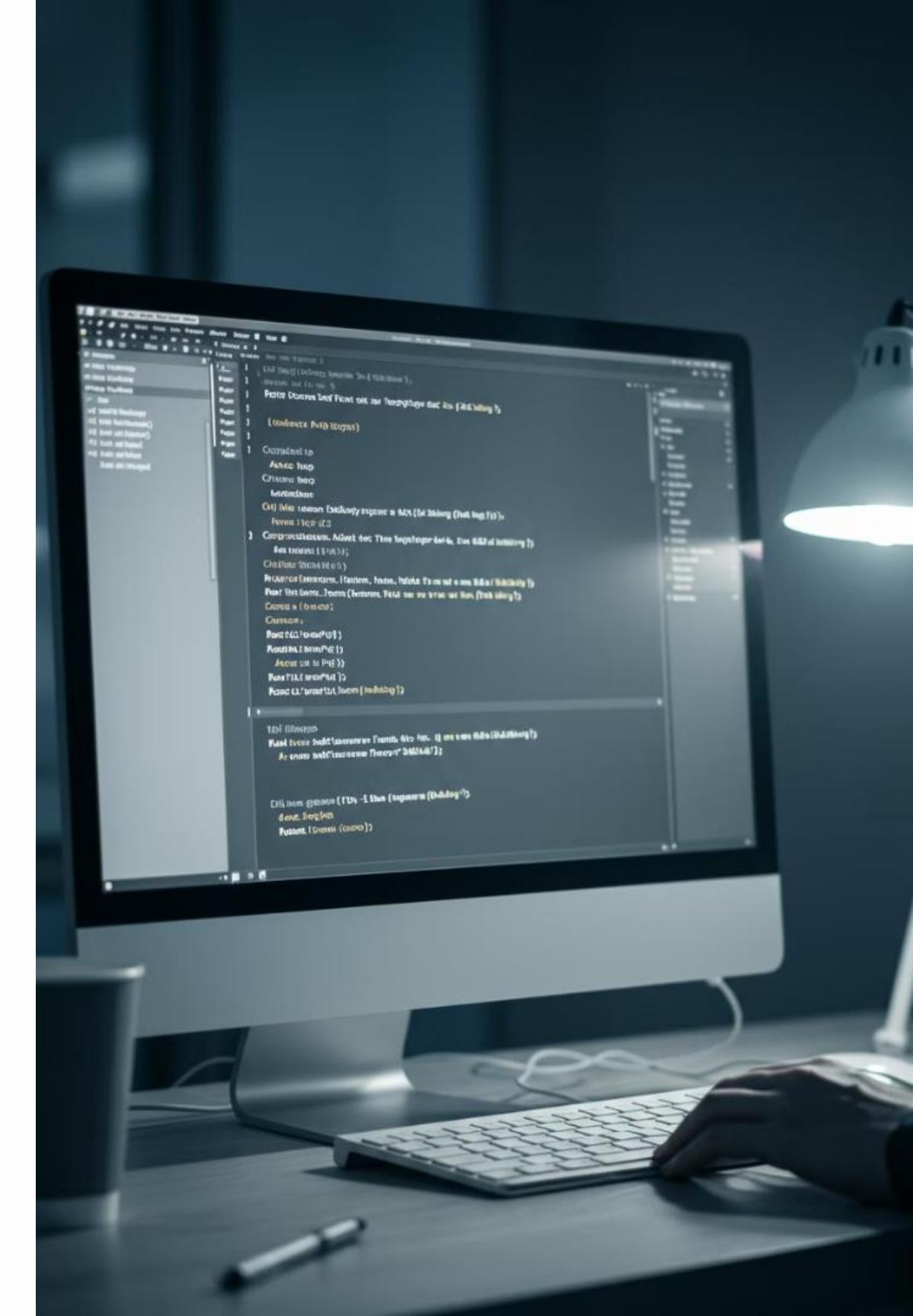
Keyword-driven prístup

Testy sa zapisujú pomocou "klúčových slov".



Čitateľný pre všetkých

Zrozumiteľný aj pre netechnických používateľov.



ROBOT FRAME WORK

Robot Framework is an open source automation framework for test automation and [robotic process automation \(RPA\)](#). It is supported by the [Robot Framework Foundation](#) and [widely used](#) in the industry.

Its [human-friendly and versatile syntax](#) uses keywords and supports [extending through libraries](#) in Python, Java, and other languages.

It integrates with other tools for comprehensive automation without licensing fees, bolstered by a rich community with hundreds of [3rd party libraries](#).

EVENTS

Filter =

Includes unsanctioned events

Find by name, location, details...

+ ADD EVENT

NEWS

[Subscribe to newsletter](#)

[March 2025 News](#)

2025-03-28



[New Blog text released!](#)

New Blog text about RoboCon 2025 recap is out!

GET STARTED

Code is worth a thousand words.

Below you'll find a live Robot Framework editor. Feel free to experiment with it! If you find bugs, please report them in [Github issues](#).

To start using Robot Framework in a project of your own, please check tabs "Install" and "Learn". Also be sure to visit the new [Robot Framework Docs](#)!

The screenshot shows a dark-themed web-based Robot Framework editor. At the top, there are three tabs: "EDITOR" (highlighted in green), "INSTALL", and "LEARN". Below the tabs are buttons for "Simple Example" (with a dropdown arrow), "Share" (with a share icon), and "Open Maximized" (with a maximize icon). To the right is a "version" dropdown set to "7.2.2" (with a dropdown arrow). The main content area is divided into sections:

- SIMPLE EXAMPLE**: Describes a test case for user login using a mocked backend API.
- Test Suite**: Describes a test suite named `TestSuite.robot` containing two test cases: "Login User with Password" and "Denied Login with Wrong Password". It mentions keyword imports from `keywords.resource` and `CustomLibrary.py`.
- Resource File**: Describes the `keywords.resource` file, mentioning keyword definitions and imports from `CustomLibrary.py`.
- Run Test**: A section at the bottom with tabs for `TestSuite.robot`, `keywords.resource`, and `CustomLibrary.py`. It includes a "RUN" button with a play icon and a code editor window showing the robot framework code for the test suite.

```
TestSuite.robot keywords.resource CustomLibrary.py RUN ►

Run Test Suite
1 *** Settings ***
2 Documentation    A test suite for valid login.
3 ...
4 ...            Keywords are imported from the resource file
5 Resource        keywords.resource
6 Default Tags   positive
7
8 *** Test Cases ***
9 Run Test
10 Login User with Password
11     Connect to Server
12     Login User      ironman      1234567890
13     Verify Valid Login  Tony Stark
14     [Teardown]    Close Server Connection
```

Oblasti použitia



Webové testy

Testovanie webových aplikácií a rozhraní.



API testovanie

Overovanie REST a SOAP API.



Databázy

Testovanie databázových operácií.



Externé systémy

Súbory, e-maily a ďalšie systémy.

V tejto časti predstavíme hlavné oblasti, kde sa Robot Framework najčastejšie využíva v praxi.

Pri webových testoch využívame predovšetkým SeleniumLibrary, ktorá umožňuje interakciu s prehliadačom, klikanie na elementy, vyplňovanie formulárov a overovanie obsahu stránok.

API testovanie je realizované cez RequestsLibrary, ktorá zabezpečuje jednoduché volanie API endpointov, odosielanie dát a validáciu odpovedí.

Pre prácu s databázami používame DatabaseLibrary, ktorá podporuje rôzne databázové systémy ako MySQL, PostgreSQL či MongoDB a umožňuje vykonávať SQL dotazy priamo z testov.

Testovanie externých systémov zahŕňa prácu so súbormi (čítanie/zápis), odosielanie a prijímanie e-mailov, alebo integráciu s inými systémami pomocou špecializovaných knižníc.



Výhody Robot Frameworku

Zrozumiteľná syntax

Prehľadný formát podobný tabuľke v súboroch .robot.

Vysoká modularita

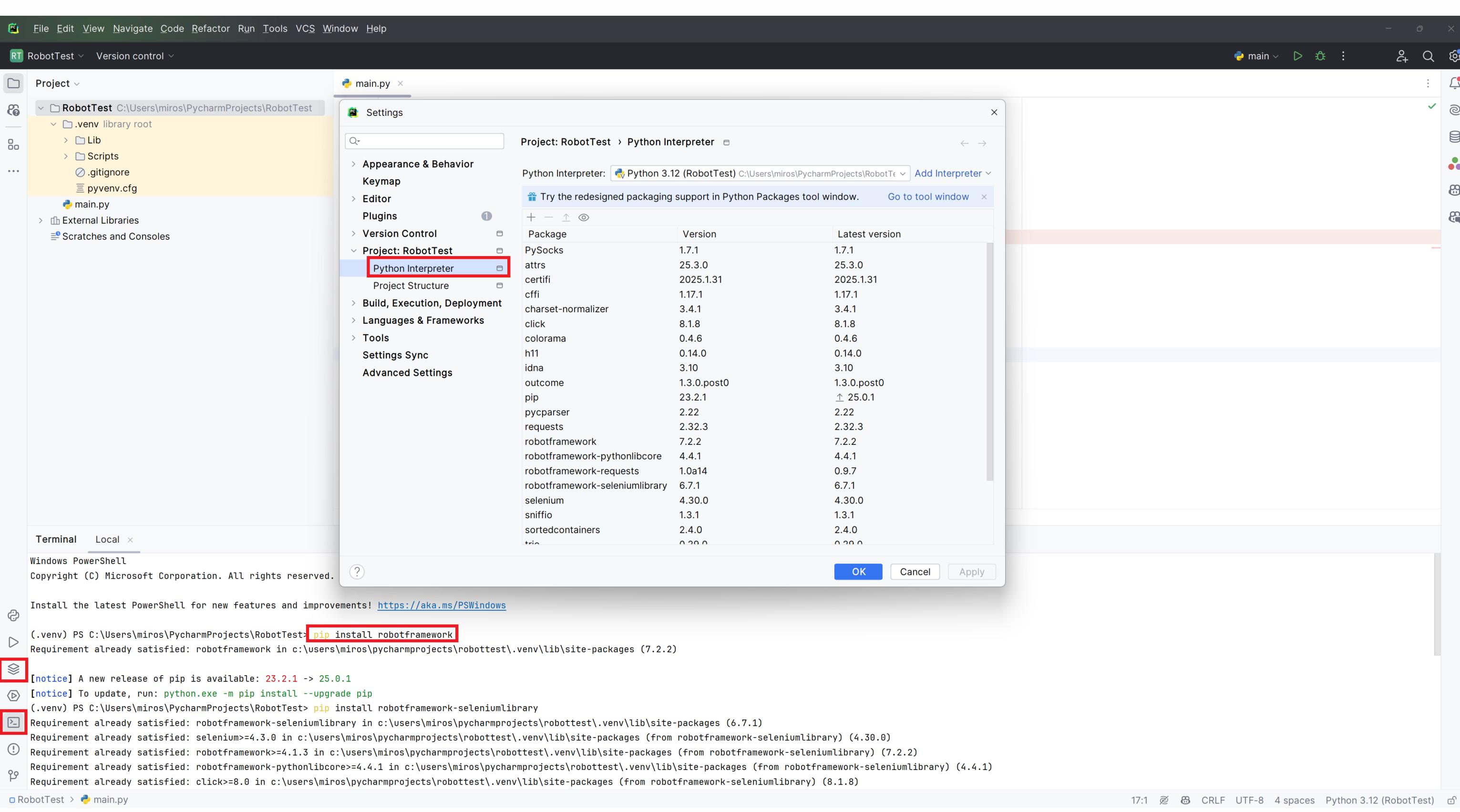
Knižnice, testovacie súbory a súbory s premennými.

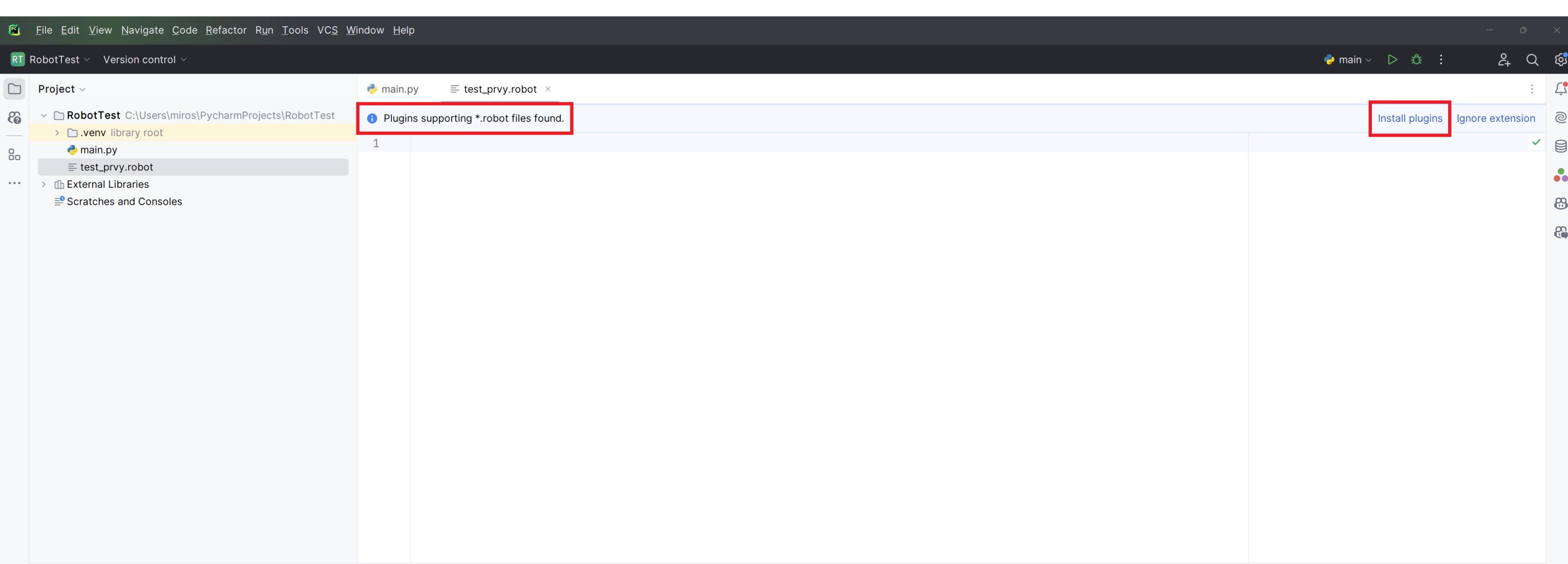
Kolaborácia

Výborné pre spoluprácu medzi QA a biznisom.

Integrácia

Prepojenie so SeleniumLibrary, RequestsLibrary a DatabaseLibrary.





Terminal Local

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

```
(.venv) PS C:\Users\miros\PycharmProjects\RobotTest> pip install robotframework
Requirement already satisfied: robotframework in c:\users\miros\pycharmprojects\robottest\.venv\lib\site-packages (7.2.2)

[notice] A new release of pip is available: 23.2.1 > 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv) PS C:\Users\miros\PycharmProjects\RobotTest> pip install robotframework-seleniumlibrary
Requirement already satisfied: robotframework-seleniumlibrary in c:\users\miros\pycharmprojects\robottest\.venv\lib\site-packages (6.7.1)
Requirement already satisfied: selenium>=4.3.0 in c:\users\miros\pycharmprojects\robottest\.venv\lib\site-packages (from robotframework-seleniumlibrary) (4.30.0)
Requirement already satisfied: robotframework>=4.1.3 in c:\users\miros\pycharmprojects\robottest\.venv\lib\site-packages (from robotframework-seleniumlibrary) (7.2.2)
Requirement already satisfied: robotframework-pythonlibcore>=4.4.1 in c:\users\miros\pycharmprojects\robottest\.venv\lib\site-packages (from robotframework-seleniumlibrary) (4.4.1)
Requirement already satisfied: click>=8.0 in c:\users\miros\pycharmprojects\robottest\.venv\lib\site-packages (from robotframework-seleniumlibrary) (8.1.8)
```

RobotTest > test_prvy.robot

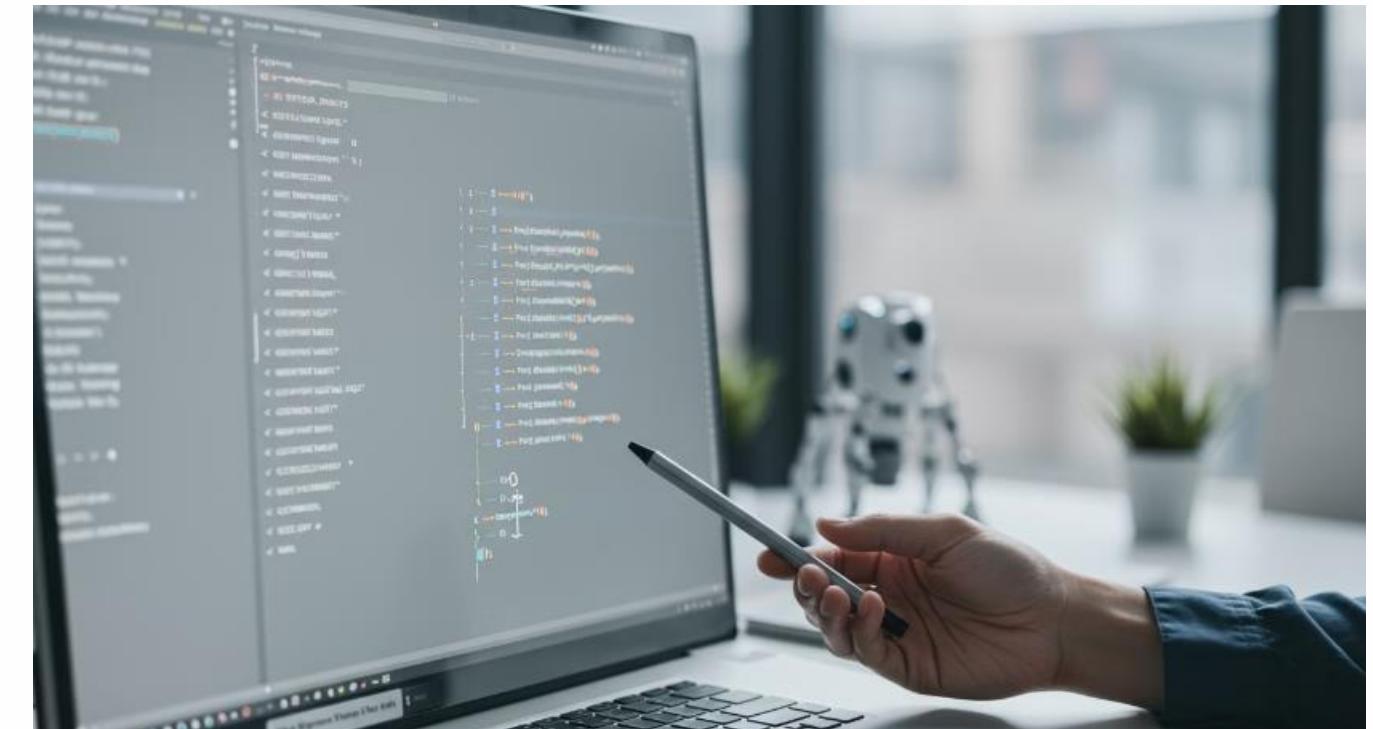
1:1 CRLF UTF-8 4 spaces Python 3.12 (RobotTest)

Základná štruktúra .robot súboru

Sekcie v .robot súbore

Robot Framework používa štruktúrovaný formát s jasne definovanými sekciami.

Každá sekcia má svoj špecifický účel v testovaní.



*** Settings ***

Import knižníc a resource súborov

*** Variables ***

Definovanie premenných

*** Test Cases ***

Hlavné testy

*** Keywords ***

Vlastné zložené kroky (funkcie)



Inštalácia Robot Framework

Inštalácia základného balíka

`pip install robotframework`

Inštalácia knižníc

```
pip install robotframework-seleniumlibrary
```

Overenie inštalácie

robot --version



Robot Framework Test - iop /Gullesec

Delone ang

Nočnú Aniš - Resubign

```
( Rexptor sarebators/ Teste ./W001)
appread teate
pfx: Feiuinay.
Pode  Passed 151 (items)
Sade  [Fes25to Mjurf;
Ende  l@perviton zotie onlagaYon rost = sate dom_jgp\vilsste prie the Lnyite tnotejd01et)
Pnde  l,| Serddoxler= Sumpo Ipste4);
Phdkeidý
Pade
Sode  Passed / Famstex
Pnde  l+ Stanke in)
File:
[passed: Test|uri
 offterea)
Executiongalv: Loumonatre tst Luste-Conn 10hudge1-wanter Flores cen/obuse Cokondi),
)
```

Scenarii -- Testy Test

Spustenie testu



Spustenie príkazu

robot testy/login.robot

Vykonanie testov

Automatické spustenie testovacích prípadov

Generovanie výstupov

Vytvorenie report.html, log.html a output.xml

Výstupy testovania



report.html

Prehľad všetkých testov s výsledkami.

The screenshot shows the Robot Framework Test Results interface. The main area displays a 'Test Overview' with various charts and graphs. Below it, there are sections for 'Test Details' and 'Test Log'. On the right side, there are several tabs and dropdown menus, including 'Test Overview', 'Test Log', 'Test Results', 'Test Details', and 'Test Summary'. The 'Test Log' tab is currently active, showing a detailed log of test steps and their outcomes.

log.html

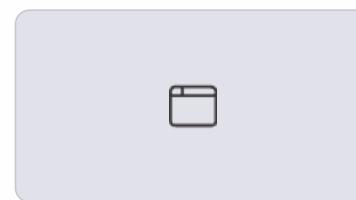
Detailné logy každého kroku v teste.

The screenshot shows a code editor displaying an XML file named 'output.xml'. The file contains test log entries in XML format. One entry shows a step to open a URL and input text into an element, followed by a key press. Another entry shows a step to click on an element with ID 'l1c1'. The code editor has syntax highlighting for XML and shows various navigation and search tools.

output.xml

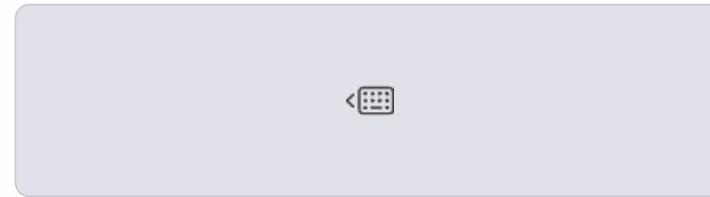
Výstup pre ďalšie spracovanie v CI.

Príklad testu – prihlásenie



Otvorenie prehliadača

Open Browser \${URL} \${BROWSER}



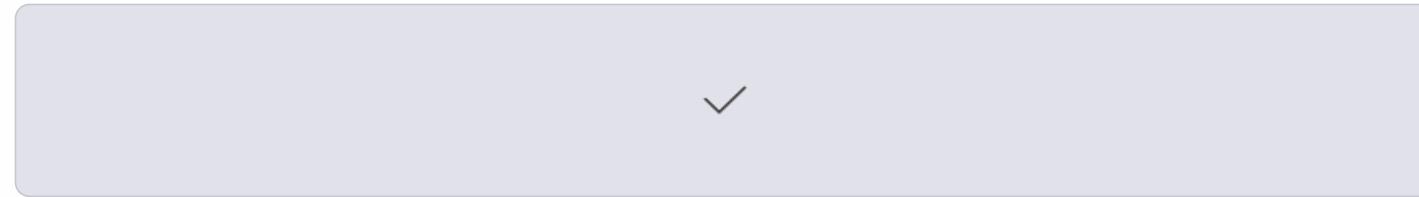
Zadanie prihlásovacích údajov

Input Text id=username \${USER}



Kliknutie na tlačidlo

Click Button id=login



Overenie prihlásenia

Page Should Contain Vitajte

Tento slajd ukazuje jednoduchý príklad Robot Framework testu pre prihlásование do systému.

Kroky v tomto príklade:

- Najprv otvoríme prehliadač s URL definovaným v premennej \${URL} pomocou prehliadača špecifikovaného v \${BROWSER}
- Zadáme prihlásovacie meno do poľa s ID "username" pomocou hodnoty z premennej \${USER}
- Klikneme na tlačidlo s ID "login" pre potvrdenie prihlásenia
- Overíme úspešné prihlásenie kontrolou, či stránka obsahuje text "Vitajte"

Všetky tieto kroky sú napísané pomocou štandardných klúčových slov (keywords) Robot Frameworku.

Klúčové pojmy

Keyword

Krok v teste, napr.
Input Text, Click
Button

Test Case

Súbor krokov
overujúcich jednu
funkcionalitu

Suite

Skupina testov

Resource

Súbor s vlastnými
keywordmi alebo
premennými



Vlastné kľúčové slová



Podpora pre REST API

RequestsLibrary

Robot Framework poskytuje knižnicu pre testovanie REST API.

Umožňuje vytvárať HTTP požiadavky a overovať odpovede.

RequestsLibrary je jednou z najpoužívanejších knižníc pre testovanie REST API v Robot Frameworku.

V príklade vidíme vytvorenie session, odoslanie GET požiadavky a overenie stavového kódu odpovede.

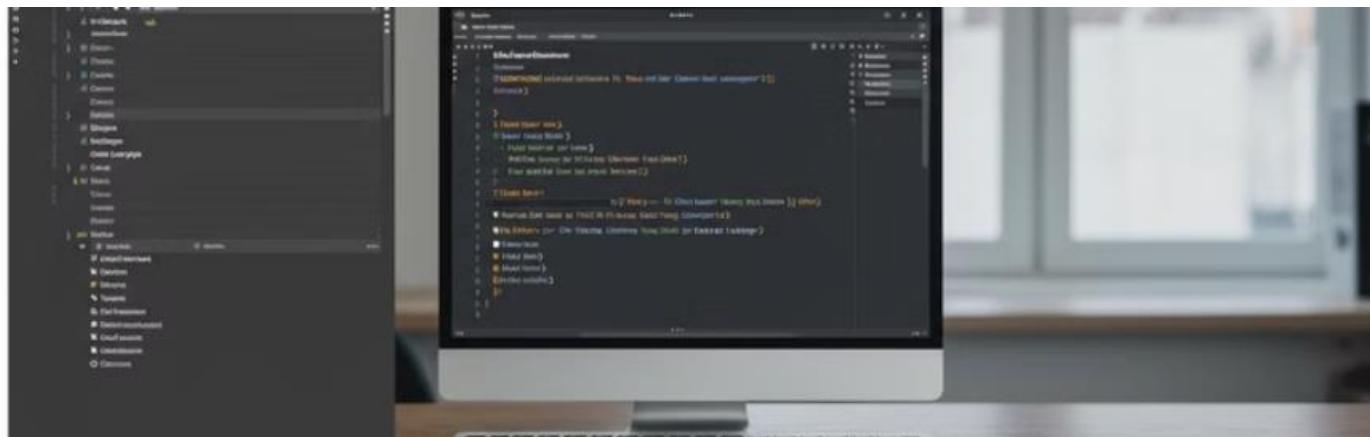
Knižnica podporuje všetky HTTP metódy (GET, POST, PUT, DELETE) a umožňuje prácu s JSON, XML a inými formátmi dát.

Pre pokročilé scenáre je možné pridať hlavičky, autentifikáciu a spracovať komplexné odpovede.

```
*** Settings ***
Library RequestsLibrary

*** Test Cases ***
Over test REST API
    Create Session moje_api https://api.test.sk
    ${response}= GET moje_api /v1/data
    Should Be Equal As Numbers
    ${response.status_code} 200
```

Integrácia



CI/CD systémy

GitHub Actions, GitLab CI, Jenkins



IDE pluginy

VSCode (Robot Framework Language Server)



Nástroje na správu testov

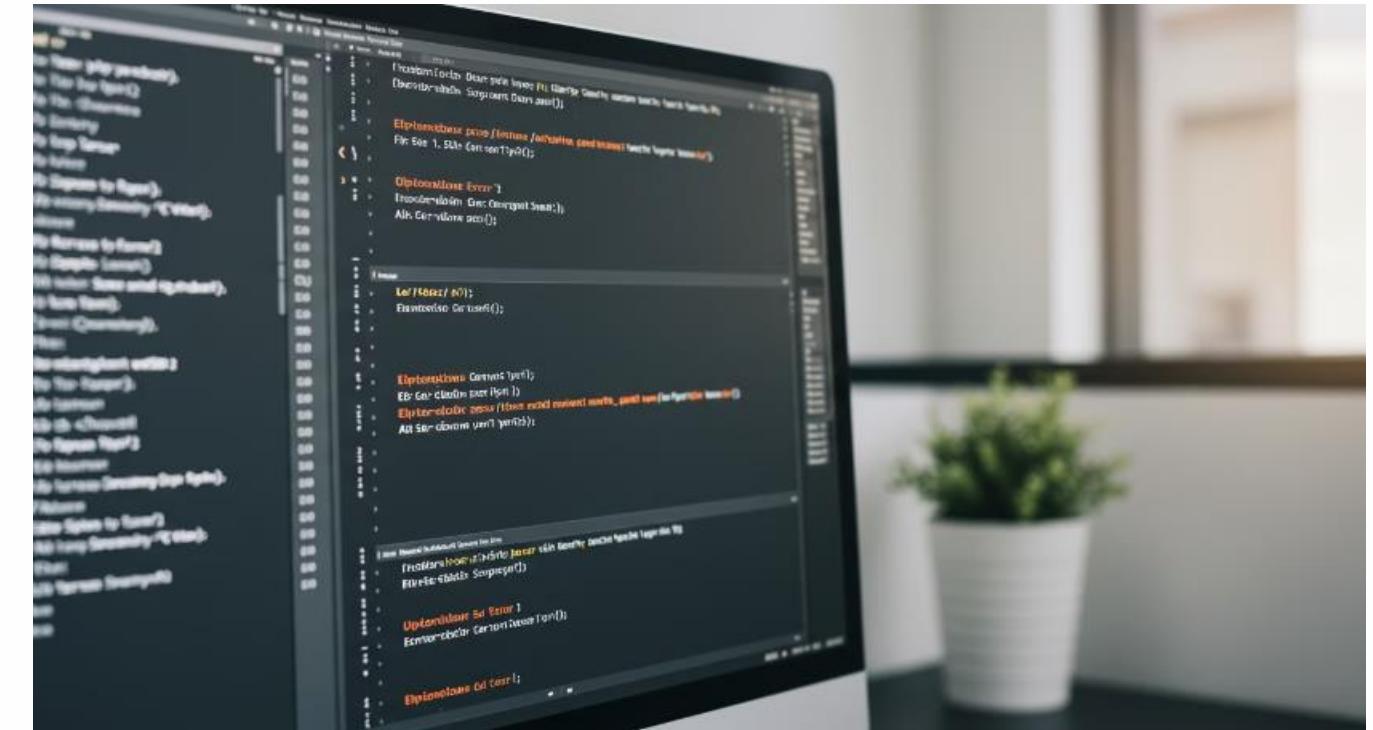
Prepojenie s TestRail a podobnými systémami

Najčastejšie chyby

Problémy pri testovaní

Pri práci s Robot Framework sa môžete stretnúť s niekoľkými typickými problémami.

Poznanie ich riešení vám ušetrí čas pri vývoji testov.



Nesprávna identifikácia prvku

Použiť id, name, xpath, css

Chýbajúce knižnice

Skontrolovať pip freeze, Library v Settings

Dlhé testy bez keywordov

Rozdeliť testy na viac menších keywordov

Nepredvídateľné správanie

Použiť Wait Until Element Is Visible

Odporúčania do praxe



Modularita

Používať vlastné keywordy na zjednodušenie logiky.



Tagovanie

Označovať testy podľa typu (napr. @critical, @smoke).



Konzistentnosť

Zachovávať jednotné pomenovanie premenných a testov.



Validácia

Využívať overovanie výstupov (Page Should Contain).

Tieto odporúčania pomáhajú zlepšiť kvalitu a udržateľnosť testovacích projektov v Robot Framework.

Modularita: Vysvetlite, ako vytvárať znovupoužiteľné keywordy, ktoré zapuzdrujú komplexnú funkcia. Ukážte príklad, ako premeniť 5-6 riadkov kódu na jeden keyword.

Tagovanie: Zdôraznite výhody používania tagov pri spúštaní konkrétnych skupín testov, napríklad: *robot -i smoke tests/* spustí len testy s tagom @smoke.

Konzistentnosť: Vysvetlite, prečo je dôležité dodržiavať jednotné konvencie pri pomenovaní - uľahčuje to orientáciu v kóde a zjednodušuje údržbu. Odporúčajte vytvoriť dokumentáciu štandardov v tíme.

Validácia: Upozornite na dôležitosť kontroly výstupov testov. Testy bez validácie môžu prejsť aj keď aplikácia nefunguje správne. Uveďte príklady rôznych validačných keywordov (Page Should Contain, Element Should Be Visible, atď.).



Testovacia architektúra

Štruktúra projektu

Dobre organizovaný projekt zlepšuje prehľadnosť a údržbu testov.

Odporúčaná štruktúra obsahuje niekoľko kľúčových adresárov.

- tests/ - hlavné test cases
- resources/ - zdieľané premenné a keywordy
- libraries/ - vlastné knižnice v Pythone
- output/ - generované reporty
- config/ - premenné pre rôzne prostredia

Pri tejto časti prezentácie vysvetlite význam správnej architektúry pre dlhodobú údržbu Robot Framework projektov.

Zdôraznite, že dodržiavanie štandardizovanej štruktúry projektu je kľúčové pre spoluprácu v tíme a zjednodušuje onboarding nových členov.

Môžete uviesť príklady problémov, ktoré vznikajú pri zlej organizácii projektu - napríklad duplicity v kóde, ťažká údržba a problémy pri rozširovaní testov.

Pripomeňte, že adresáre možno prispôsobiť potrebám konkrétneho projektu, ale základná štruktúra by mala zostať konzistentná.

Opäťovné použitie pomocou Resource Files



Resource súbory sú kľúčovou súčasťou efektívneho testovania v Robot Framework. Umožňujú nám vytvárať znovupoužiteľný kód.

V prvom kroku vytvárame súbor s keywordami, ktoré budeme používať vo viacerých testoch. Tieto keywordy môžu obsahovať komplexné postupy ako prihlásenie, navigáciu, alebo validáciu dát.

V druhom kroku importujeme resource súbor do testov pomocou kľúčového slova `Resource` a relatívnej cesty k súboru.

V treťom kroku používame definované keywordy priamo v testoch, čo robí testy prehľadnejšie a stručnejšie.

Štvrtý krok predstavuje princíp DRY (Don't Repeat Yourself), ktorý je základom efektívneho programovania a testovania. Zamedzuje duplikácii kódu a zjednoduší údržbu.

The screenshot shows the Stulpation software interface. At the top, there's a navigation bar with tabs: Unfinished, Reproducible, Noncritical, and Report. Below the navigation is a sidebar with categories: 1 Testers, Performance, Security, Security+, Security, Verifying, Accessibility, Acceptability, Usability, Usability, Usability, Usability, Usability, and Usability. The main area displays a grid of test cards. A large section titled "Your priority" lists tests: Test 1 (Test), Test 2 (Test), Test 3 (Test), Test 4 (Test), Test 5 (Test), Test 6 (Test), Test 7 (Test), Test 8 (Test), Test 9 (Test), Test 10 (Test), Test 11 (Test), Test 12 (Test), Test 13 (Test), Test 14 (Test), Test 15 (Test), Test 16 (Test), Test 17 (Test), Test 18 (Test), Test 19 (Test), and Test 20 (Test). Each card includes a title, a brief description, and a status indicator.

Používanie tagov



Označenie testov

[Tags] critical regression

Filtrovanie pri spustení

robot --include critical tests/

Vylúčenie testov

robot --exclude beta tests/

Organizácia testov

Logické zoskupovanie podľa účelu



Dynamické údaje: RandomLibrary, FakerLibrary

100+

Typov generovaných dát

Mená, e-maily, adresy, telefónne čísla a
ďalšie.

10X

Rýchlejšie testovanie

Bez potreby manuálneho vytvárania
testovacích dát.

99%

Pokrytie hraničných prípadov

Testovanie s rôznymi typmi vstupných
dát.

Tvorba vlastnej knižnice v Pythone

Rozšírenie možností

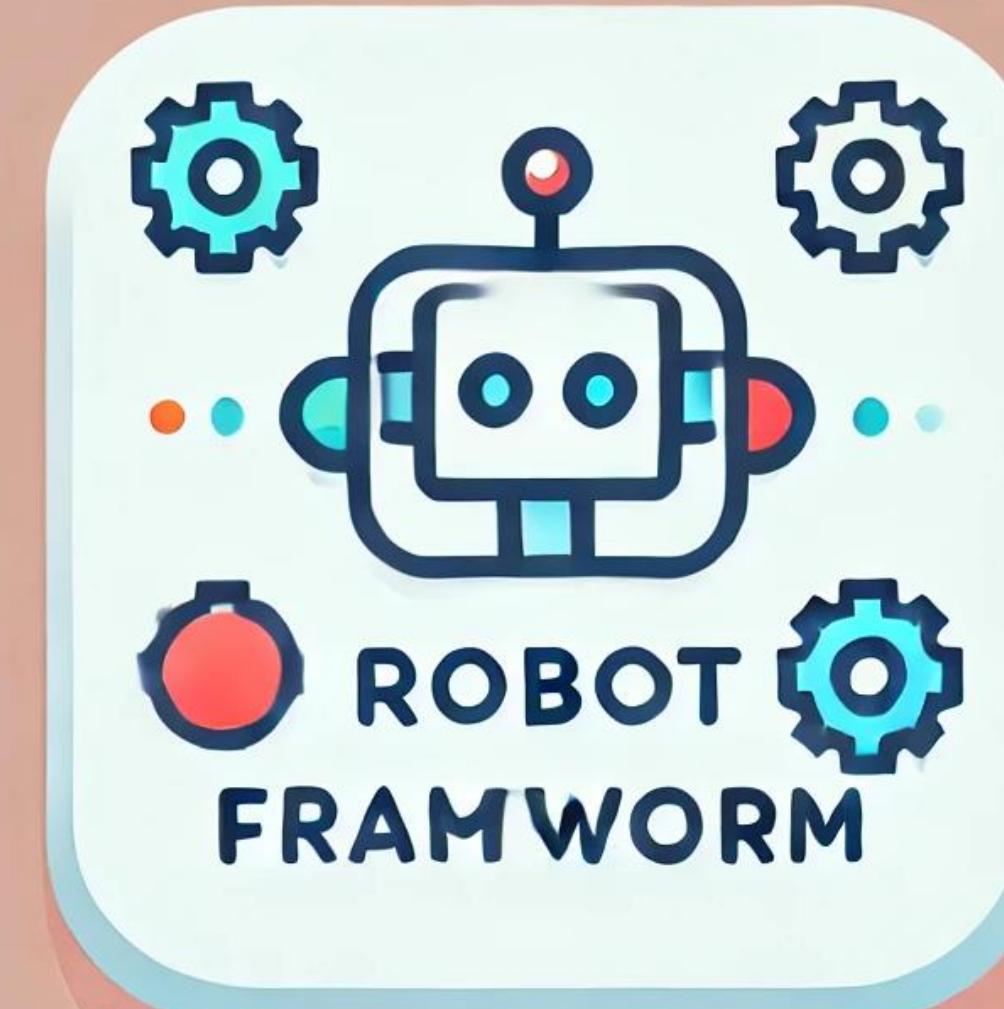
Robot Framework umožňuje vytvárať vlastné knižnice v Pythone.

Tie poskytujú prístup k zložitej logike, databázam či API.

```
class CustomLibrary:  
    def pozdrav(self, meno):  
        return f"Ahoj, {meno}!"  
  
# V .robot súbore:  
*** Settings ***  
Library ../libraries/custom_library.py  
  
*** Test Cases ***  
Test Pozdrav  
    ${msg}= Pozdrav Miroslav  
    Should Be Equal ${msg} Ahoj, Miroslav!
```

Úlohy Robot FW

1. Čo je to Robot Framework?
2. Ako sa nainštaluje a používa?
3. Ako sa v ňom tvoria testy?
4. Ako vyzerá základná štruktúra?
5. Čo sú to resources/variables?
6. Aké sú vhodné prípady použitia?
7. Kde nájdeme dokumentáciu?



Ako začať s Testovaním v Pytest

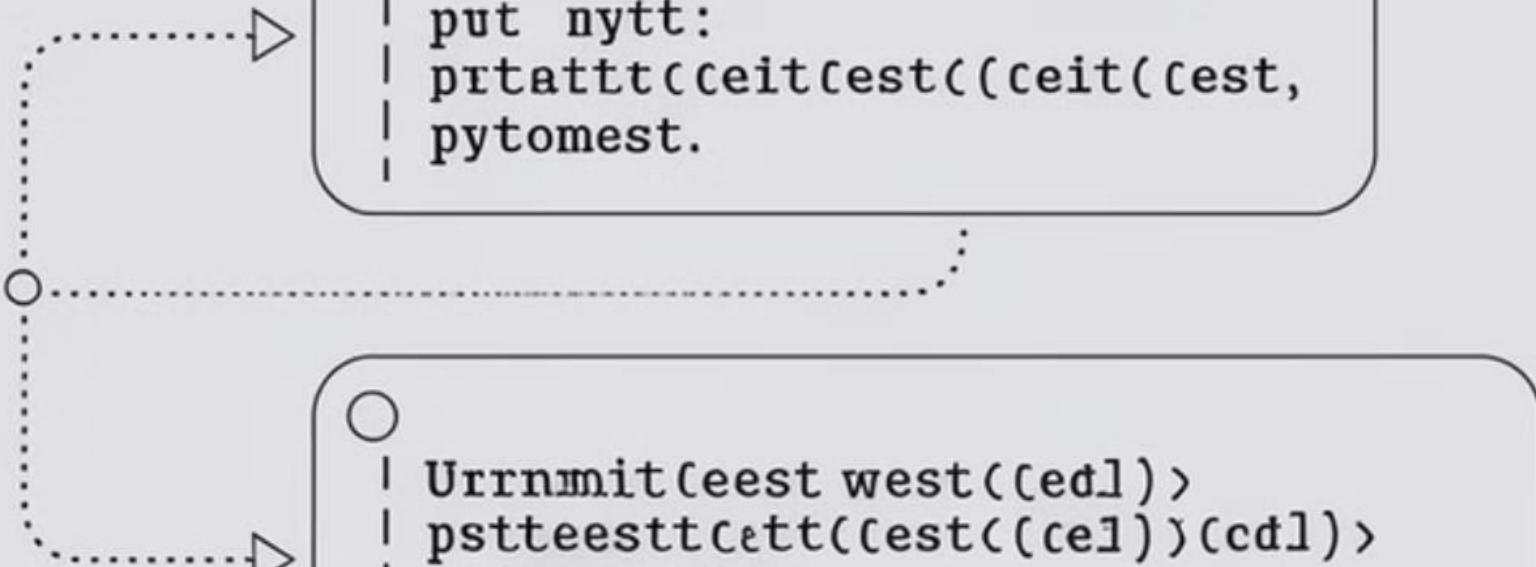
AKREDITOVANÝ KURZ





Čo sa naučíme?

1. Čo je to Pytest?
2. Ako sa integruje a používa?
3. Aké existujú testy?
4. Ako vyzerá základná štruktúra?
5. Aké má komponenty?
6. Aké sú vhodné prípady použitia
7. Kde nájdeme dokumentáciu?
8. Profit



Automatizácia testov s pytest

Jednoduchý zápis

Bez nutnosti definovať triedy.

Prirodzený syntax

Testovacie funkcie vyzerajú ako obyčajné Python funkcie.

Čitateľný výstup

Veľmi čitateľný výstup pri chybách.

Rozšíritelnosť

Rozsiahla knižnica pluginov (napr. pytest-cov, pytest-django).

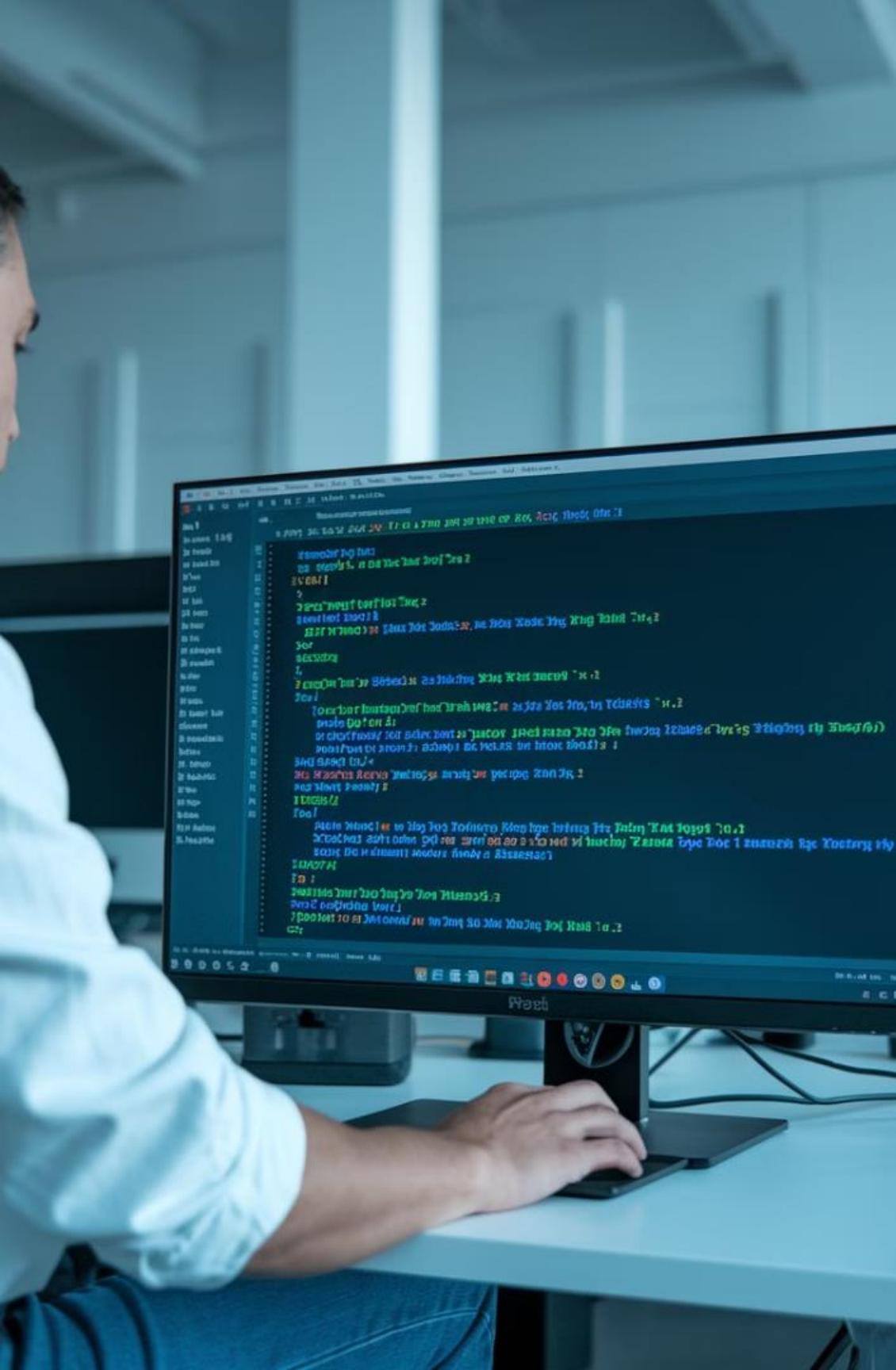
Výhody pytest oproti unittest

Jednoduchosť

- Bez tried a dedičnosti
- Prirodzené assert výrazy
- Menej kódu, viac čitateľnosti

Pokročilé funkcie

- Fixtures pre zdieľané dátá
- Parametrizácia testov
- Pluginy pre rozšírenie funkcionality
- Kompatibilita s unittest

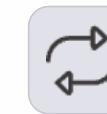


Kedy je pytest vhodné použiť?



Stredné a veľké projekty

Ideálny pre stredne veľké a veľké Python projekty.



Opakované spúšťanie

Ak sa vyžaduje opakované spúšťanie testov.



Zdieľané prostredie

Ak testy potrebujú zdieľať prípravu prostredia.



Automatizácia

Pri automatickom spúšťaní v CI (napr. GitHub Actions).

Inštalácia a nastavenie pytest

Inštalácia cez pip

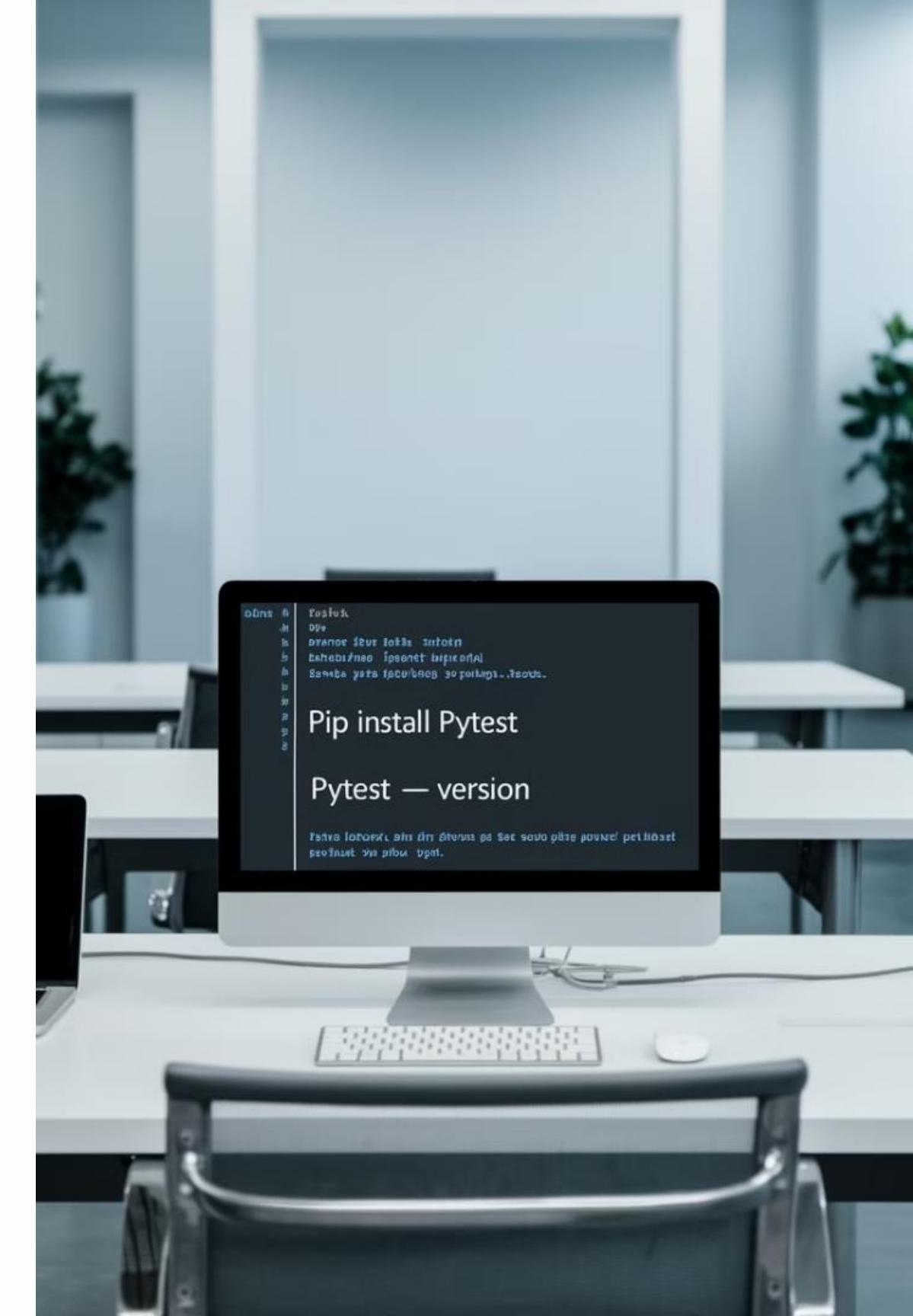
```
pip install pytest
```

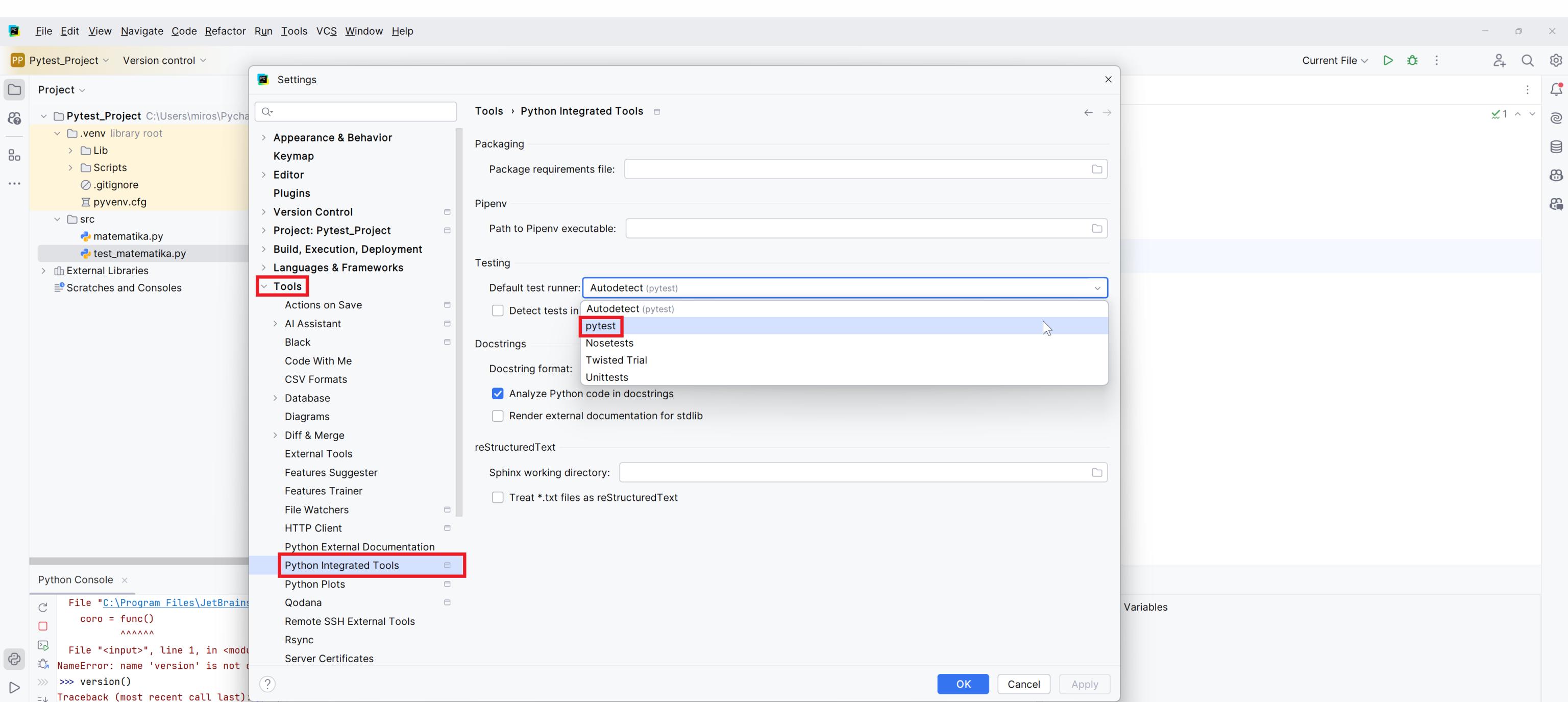
Overenie inštalácie

```
pytest --version
```

Voliteľná konfigurácia

```
[pytest]
minversion = 7.0
addopts = -ra -q --tb=short
testpaths = tests
```





Ctrl + Shift + F10

The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Project Bar:** Pytest_Project, Version control.
- Toolbars:** Standard toolbar with icons for file operations.
- Code Editor:** The current file is `test_matematika.py`. The code contains:

```
from src.matematika import sucet
def test_sucet():
    assert sucet(a: 2, b: 3) = 5
```
- Contextual Menu:** A context menu is open over the word "sucet" in the third line of code. It includes options like New, Cut, Copy, Paste, Find Usages, Rename..., Refactor, Bookmarks, Reformat Code, Optimize Imports, Delete..., Override File Type, and Run 'pytest in test_matematika.py' (highlighted with a red box).
- Run Tab:** Shows the "Test" section with the following items:
 - Run 'pytest in test_matematika.py'
 - Debug 'pytest in test_matematika.py'
 - More Run/Debug
 - Open in Right Split
 - Open In
 - Local History
 - Repair IDE on File
 - Reload from Disk
 - Compare With...
 - Diagrams
 - Convert to Jupyter Notebook
 - GitHub Copilot
- Output Log:** Displays the terminal output of the pytest run:

```
Tests passed: 1 of 1 test - 0 ms
Testing started at 8:03 ...
Launching pytest with arguments C:/Users/miros/PycharmProjects/Pytest_Project/src/test_matematika.py --no-header --no-summary -q in C:/Users/miros/PycharmProjects/Pytest_Project/src
=====
test session starts =====
collecting ... collected 1 item
test_matematika.py::test_sucet PASSED [100%]
=====
1 passed in 0.02s =====
Process finished with exit code 0
```
- Status Bar:** Pytest_Project > src > test_matematika.py, 5:1, CRLF, UTF-8, 4 spaces, Python 3.12 (Pytest_Project).

The screenshot shows the PyCharm IDE interface with the following details:

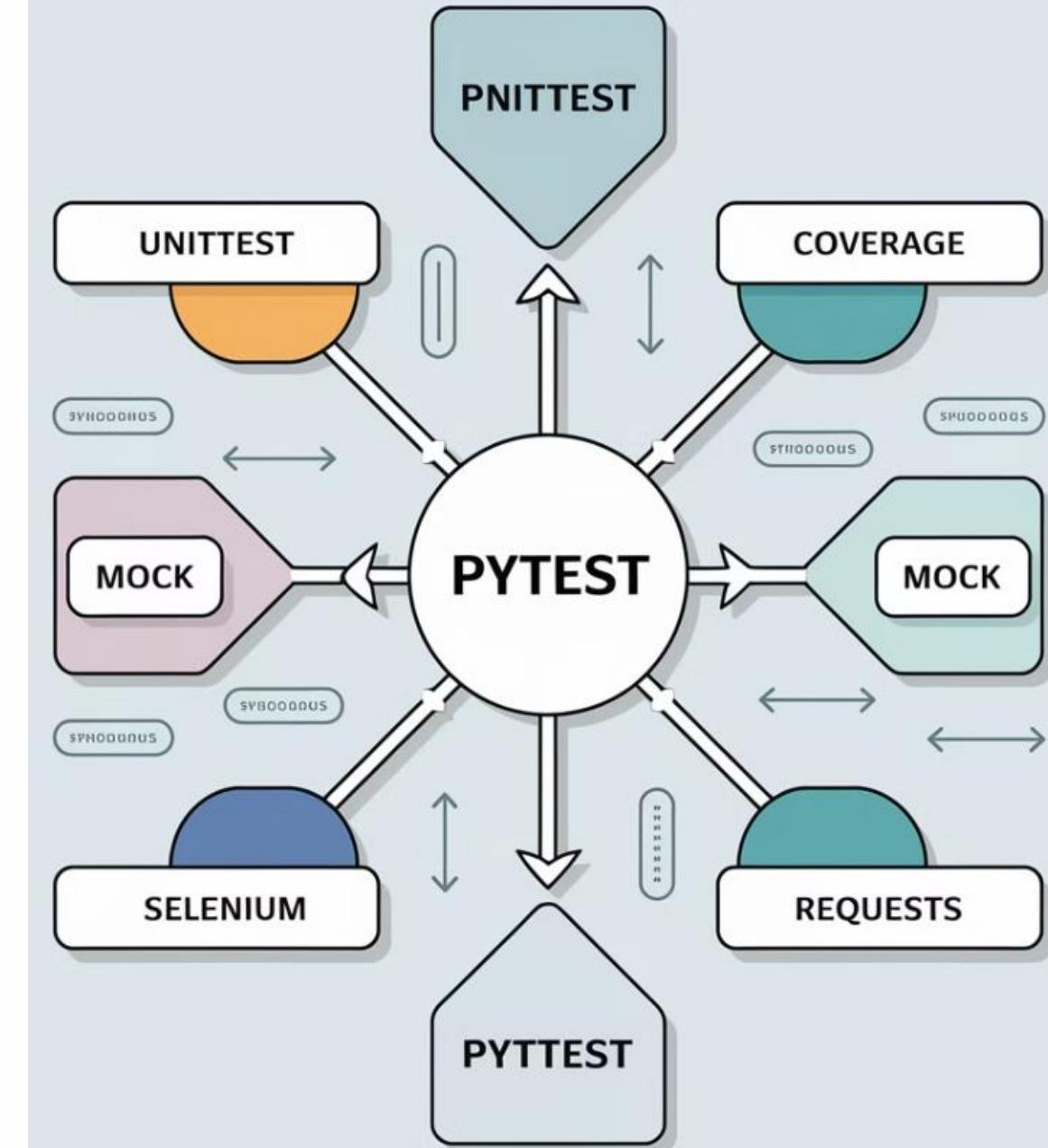
- File Bar:** File Edit View Navigate Code Refactor Run Tools VCS Window Help
- Project Bar:** Pytest_Project Version control
- Toolbars:** Standard toolbar icons.
- Left Sidebar:** Project tree showing Pytest_Project structure. The `tests` folder and `pytest.ini` file are highlighted with red boxes.
- Central Editor:** Content pane showing the `pytest.ini` file with the following content:

```
[pytest]
pythonpath = .
```
- Bottom Terminal:** Shows the command `pytest -v` being run, resulting in:

```
===== 1 passed in 0.01s =====
(.venv) PS C:\Users\miros\PycharmProjects\Pytest_Project> pytest -v
===== test session starts =====
platform win32 -- Python 3.12.2, pytest-8.3.5, pluggy-1.5.0 -- C:\Users\miros\PycharmProjects\Pytest_Project\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\miros\PycharmProjects\Pytest_Project
configfile: pytest.ini
collected 1 item
tests/test_matematika.py::test_sucet PASSED
===== 1 passed in 0.01s =====
(.venv) PS C:\Users\miros\PycharmProjects\Pytest_Project>
```
- Bottom Status Bar:** Pytest_Project > pytest.ini, 3:1, CRLF, UTF-8, 4 spaces, Python 3.12 (Pytest_Project)

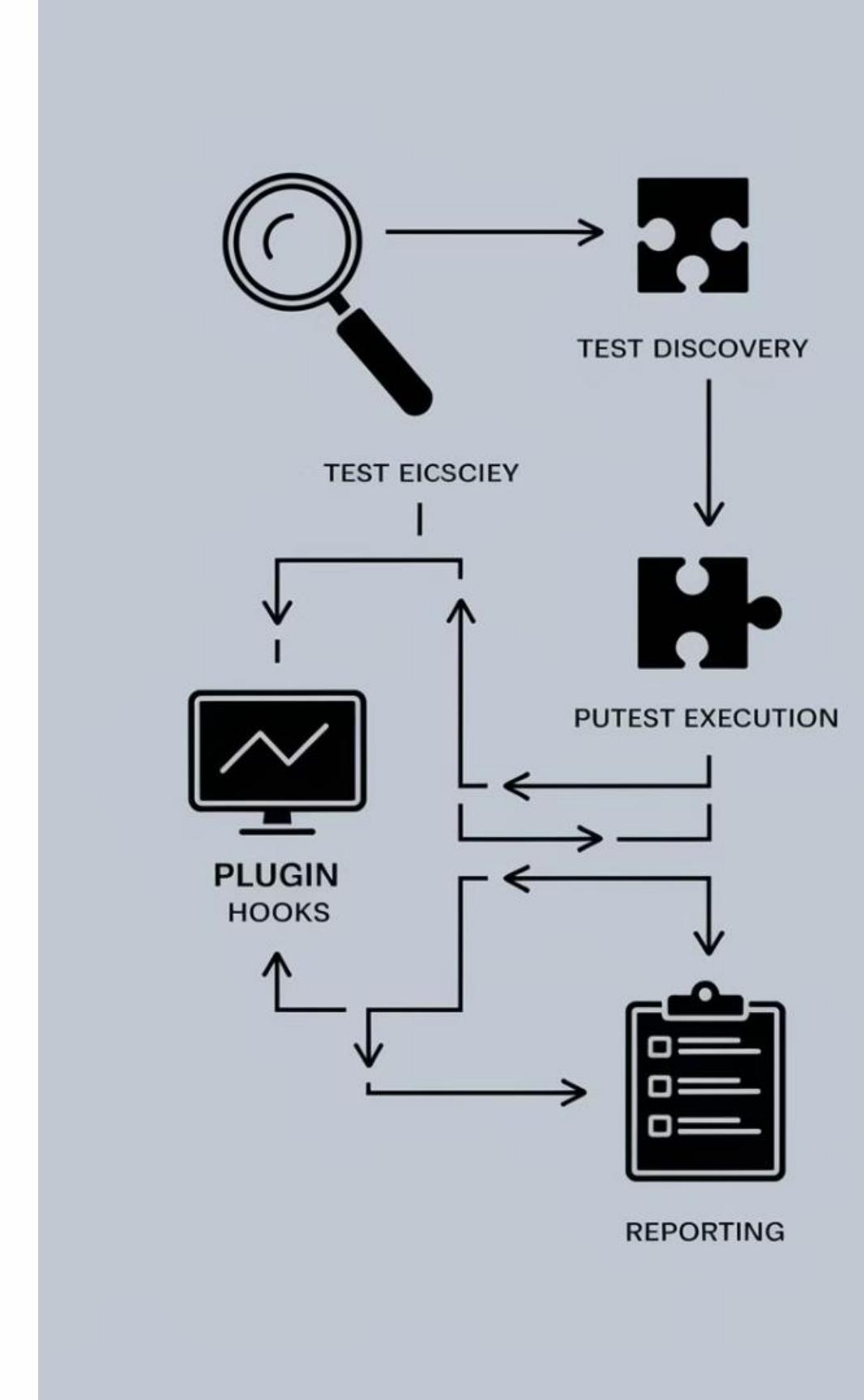
Odporúčané balíčky pre pytest

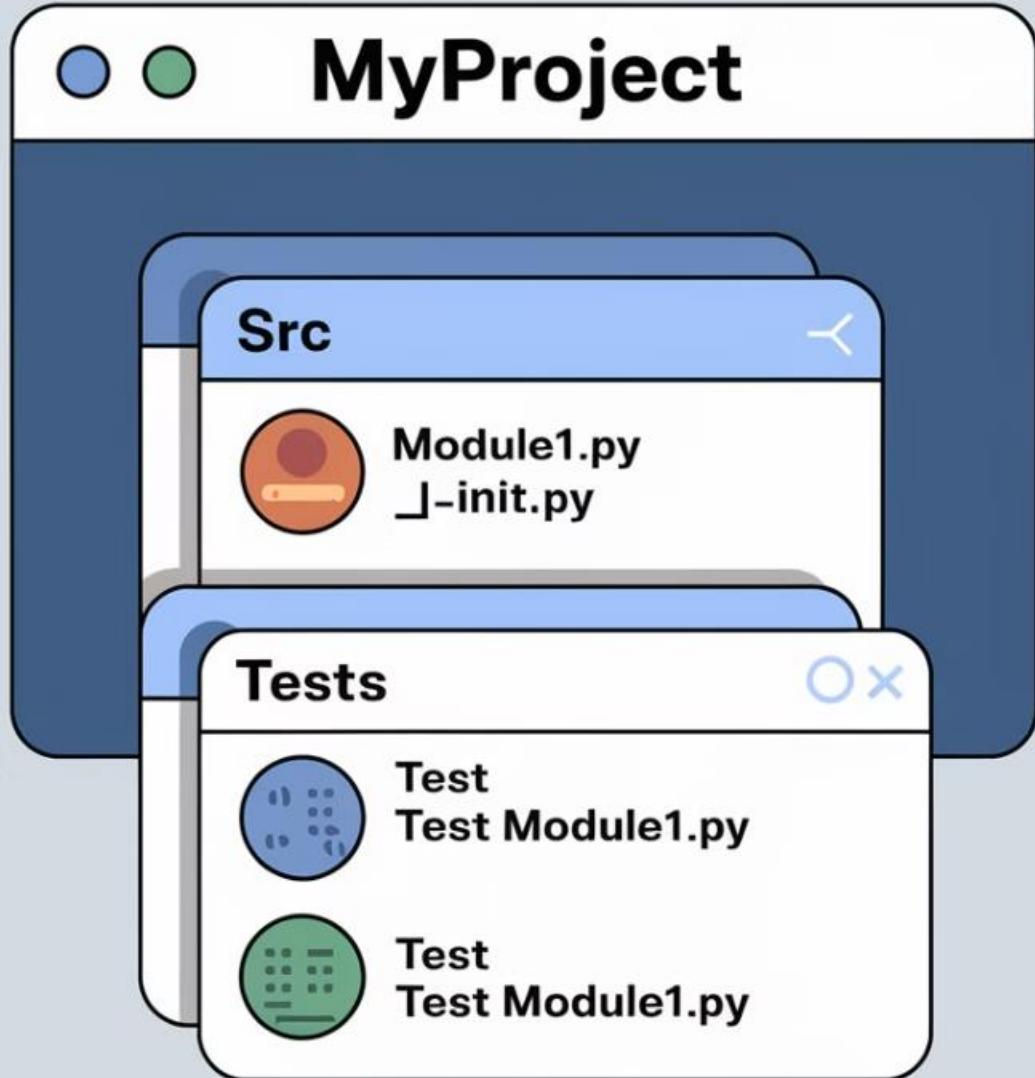
| Balík | Účel |
|-----------------------------|------------------------------|
| pytest-cov | meranie pokrytie kódu |
| pytest-xdist | paralelné spúšťanie testov |
| pytest-mock | mockovanie a patchovanie |
| pytest-django, pytest-flask | integrácia s web frameworkmi |



Základné komponenty pytest

| Komponent | Popis |
|--------------------|---|
| Testovacie funkcie | Funkcie začínajúce test_ |
| Fixtures | Preddefinované objekty dostupné ako argumenty testov |
| Parametrize | Spúšťa testy s rôznymi vstupmi |
| Hooks | Prístup k interným udalostiam (napr. pred/po testovaní) |
| Plugins | Rozšírenie funkcionality bez úpravy jadra |





Štruktúra adresárov pre pytest

```
projekt/
|
|   src/
|   |
|   |   kalkulacka.py
|
|   tests/
|   |
|   |   test_kalkulacka.py
```

Testovacie súbory začínajú prefixom `test_`, testovacie funkcie začínajú na `test_`. Pytest automaticky deteguje všetky testy.

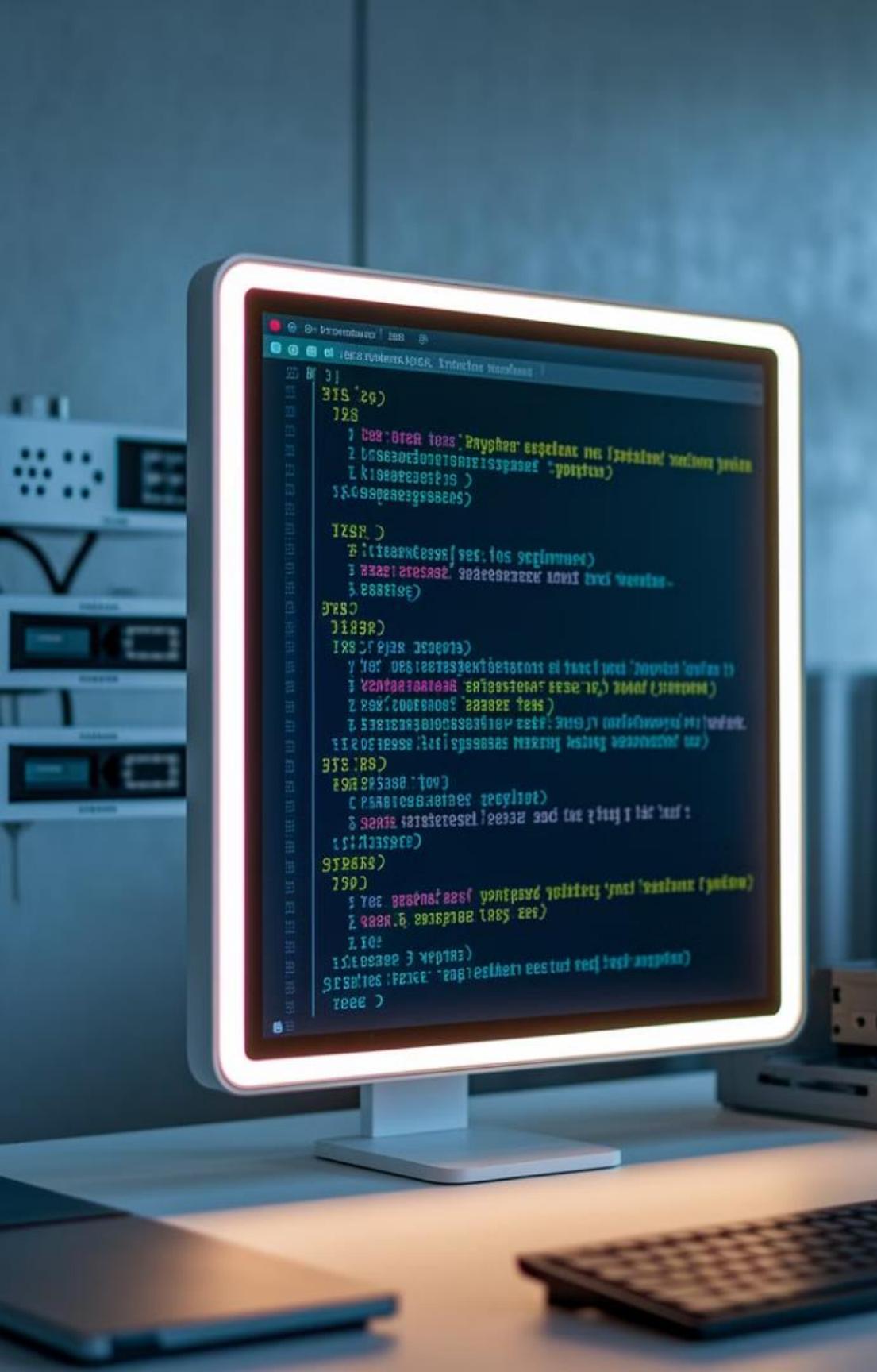
Písanie jednoduchých testov pomocou pytest

Testovaný kód (src/kalkulacka.py)

```
def sucet(a, b):  
    return a + b  
  
def podiel(a, b):  
    return a / b
```

Testovací súbor (tests/test_kalkulacka.py)

```
from src.kalkulacka import sucet, podiel  
  
def test_sucet():  
    assert sucet(2, 3) == 5  
    assert sucet(-1, 1) == 0  
  
def test_podiela():  
    assert podiel(10, 2) == 5
```

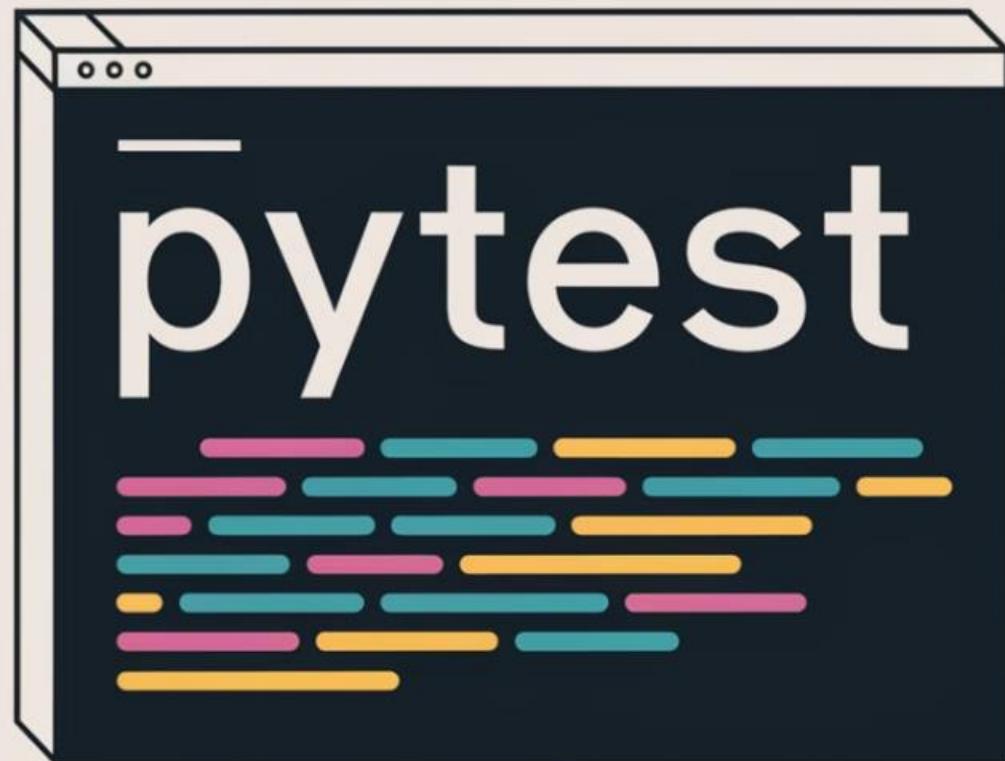


Test očakávajúci výnimku v pytest

```
import pytest

def test_delenie_nulou():
    with pytest.raises(ZeroDivisionError):
        podiel(1, 0)
```

Test je úspešný, ak kód vo with bloku vyhodí očakávanú výnimku ZeroDivisionError.



Spúšťanie testov s pytest

1

Spustiť všetky testy

```
pytest
```

2

Spustiť konkrétny
súbor

```
pytest  
tests/test_kalkulack  
a.py
```

3

Spustiť konkrétnu funkciu

```
pytest tests/test_kalkulacka.py::test_sucet
```

Výstup z pytest – prehľad

Úspešné testy

```
collected 3 items  
tests/test_kalkulacka.py ... [100%]
```

Neúspešný test

```
> assert sucet(2, 3) == 6  
E assert 5 == 6
```

pytest zvýrazní rozdiely a poskytne presné miesto zlyhania.



Zhrnutie výhod pytest



Jednoduchý zápis

Minimum šablón,
prirodzený Python
syntax.



Rýchle vykonávanie

Efektívne spúšťanie
testov s možnosťou
paralelizácie.



Zrozumiteľný výstup

Detailné informácie o
zlyhaní testov.

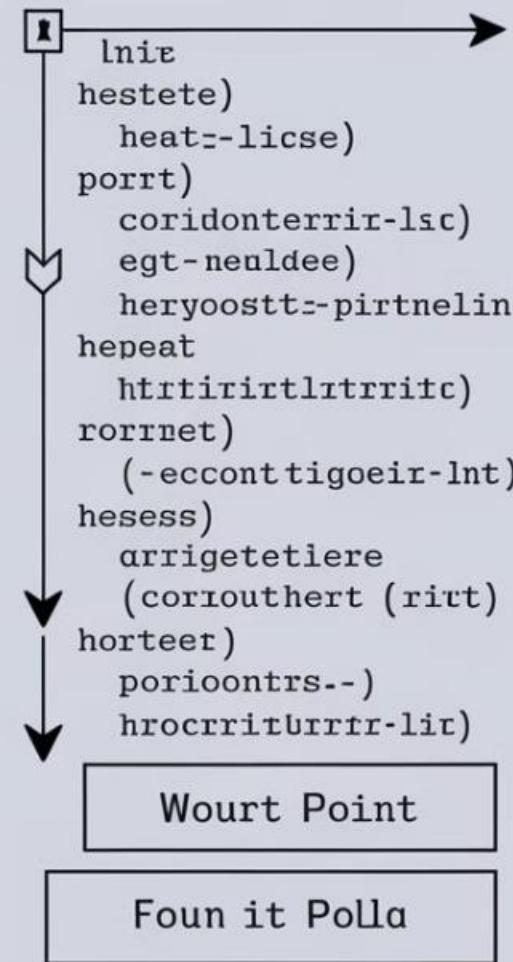
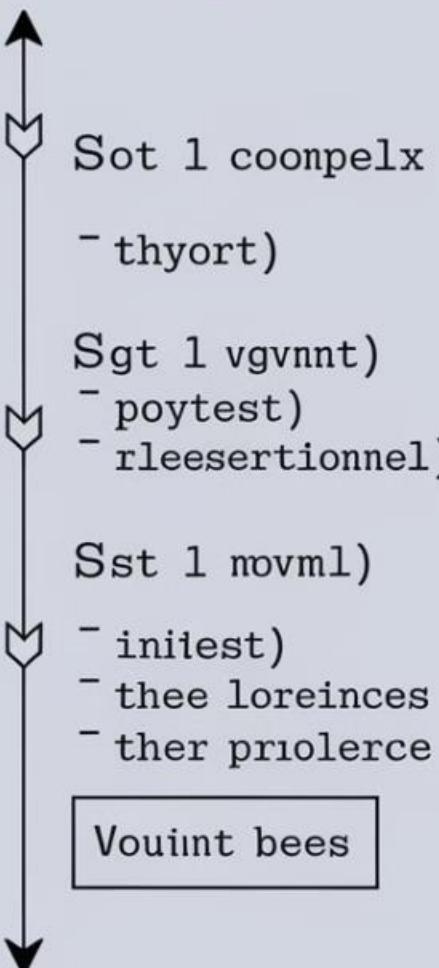


Výkonné nástroje

Fixtures, hooks,
parametrizácia a
plugins.

PYTEST

UNITTEST



Testy ako obyčajné funkcie

```
def test_max_cislo():
    assert max([1, 5, 3]) == 5
```

pytest detektuje funkcie začínajúce na `test_`. Nie je potrebné dediť triedu ani volať `unittest.main()`.

@pytest.mark.parametrize
@pytest.parametrize

Parameterizácia testov

```
import pytest

@pytest.mark.parametrize("a,b,vysledok", [
    (2, 3, 5),
    (-1, 1, 0),
    (0, 0, 0)
])
def test_sucet(a, b, vysledok):
    assert a + b == vysledok
```

Test sa spustí 3x s rôznymi dátami. Pytest zobrazuje všetky varianty ako samostatné testy.

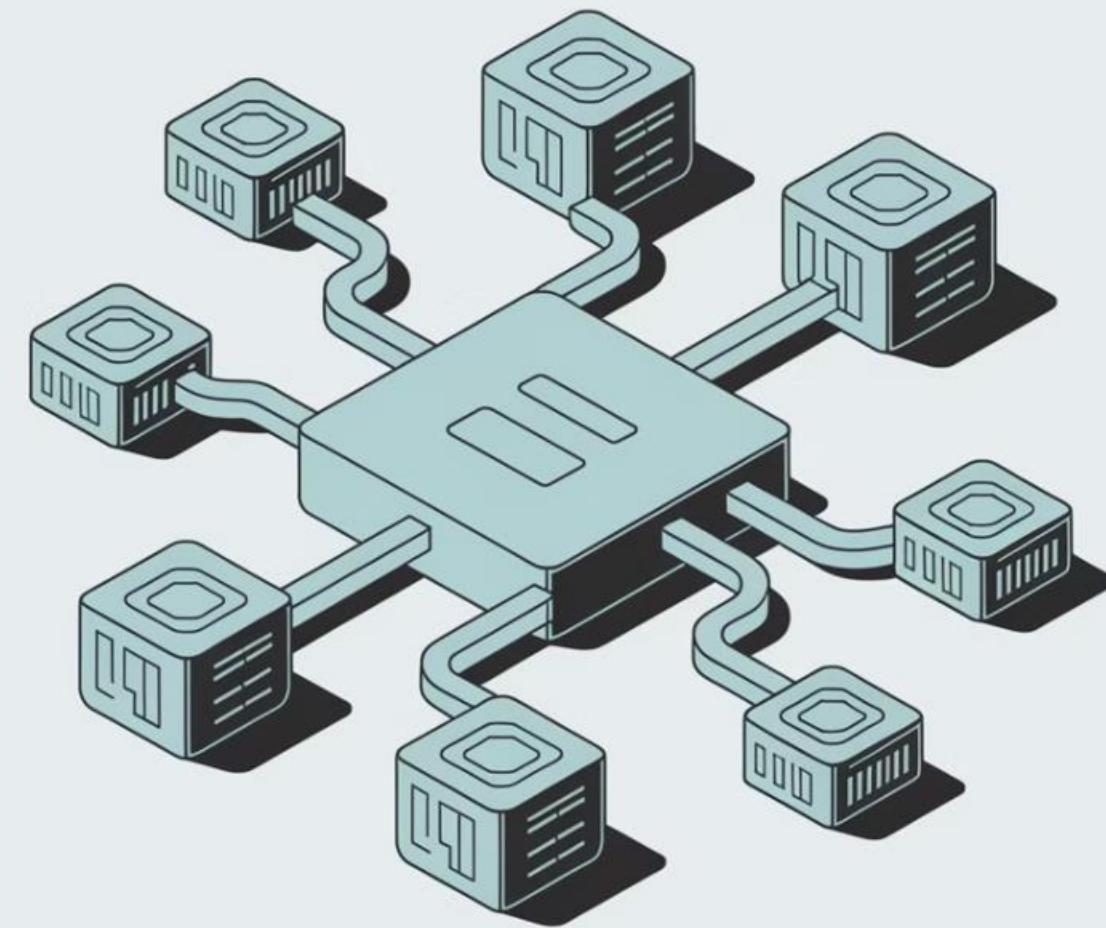
Fixtures – opakovateľné testovacie dáta

```
import pytest

@pytest.fixture
def vzorka_dat():
    return [1, 2, 3, 4]

def test_max_vzorka(vzorka_dat):
    assert max(vzorka_dat) == 4
```

Fixtures sú funkcie, ktoré sa automaticky vykonajú pred testom a dodajú mu vstup.



Písanie robustných testov

Modul: matematika.py

```
def sucet(a, b):
    return a + b

def rozdiel(a, b):
    return a - b

def podiel(a, b):
    if b == 0:
        raise ZeroDivisionError("Delenie nulou nie je
povolené")
    return a / b
```

Testy: test_matematika.py

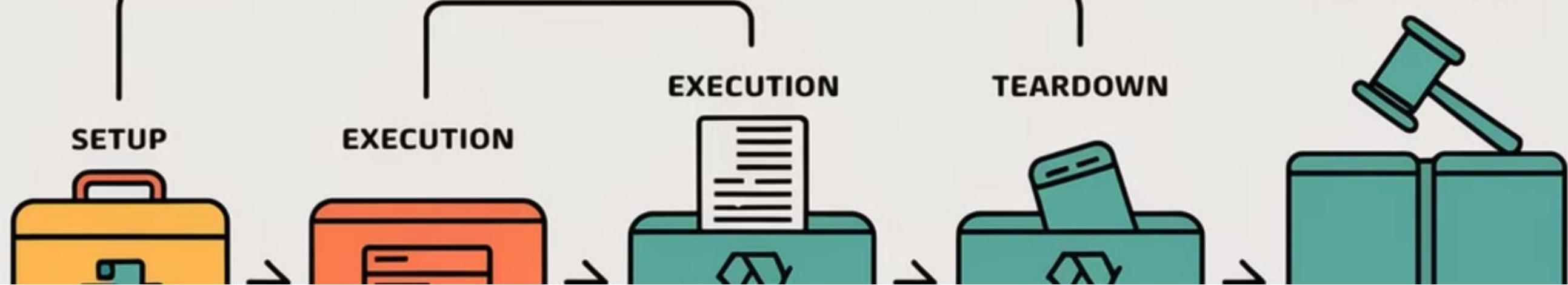
```
import pytest
from src.matematika import sucet, rozdiel, podiel

@pytest.mark.parametrize("a,b,vysledok", [(1, 2, 3), (-1, 1, 0), (0, 0, 0)])
def test_sucet(a, b, vysledok):
    assert sucet(a, b) == vysledok

def test_rozdiel():
    assert rozdiel(5, 3) == 2

def test_podiela():
    assert podiel(10, 2) == 5

def test_delenie_nulou():
    with pytest.raises(ZeroDivisionError, match="Delenie
nulou"):
        podiel(1, 0)
```



Fixtures s rôznymi scope

```
@pytest.fixture(scope="module")
def cisla():
    print("Vytváram dátá")
    data = [2, 4, 6]
    yield data
    print("Čistím dátá")
    data.clear()

def test_porne(cisla):
    for x in cisla:
        assert x % 2 == 0
```

Fixtures môžu mať rôzne scopes: function, module, session, class. Podpora autouse=True pre implicitné spustenie.

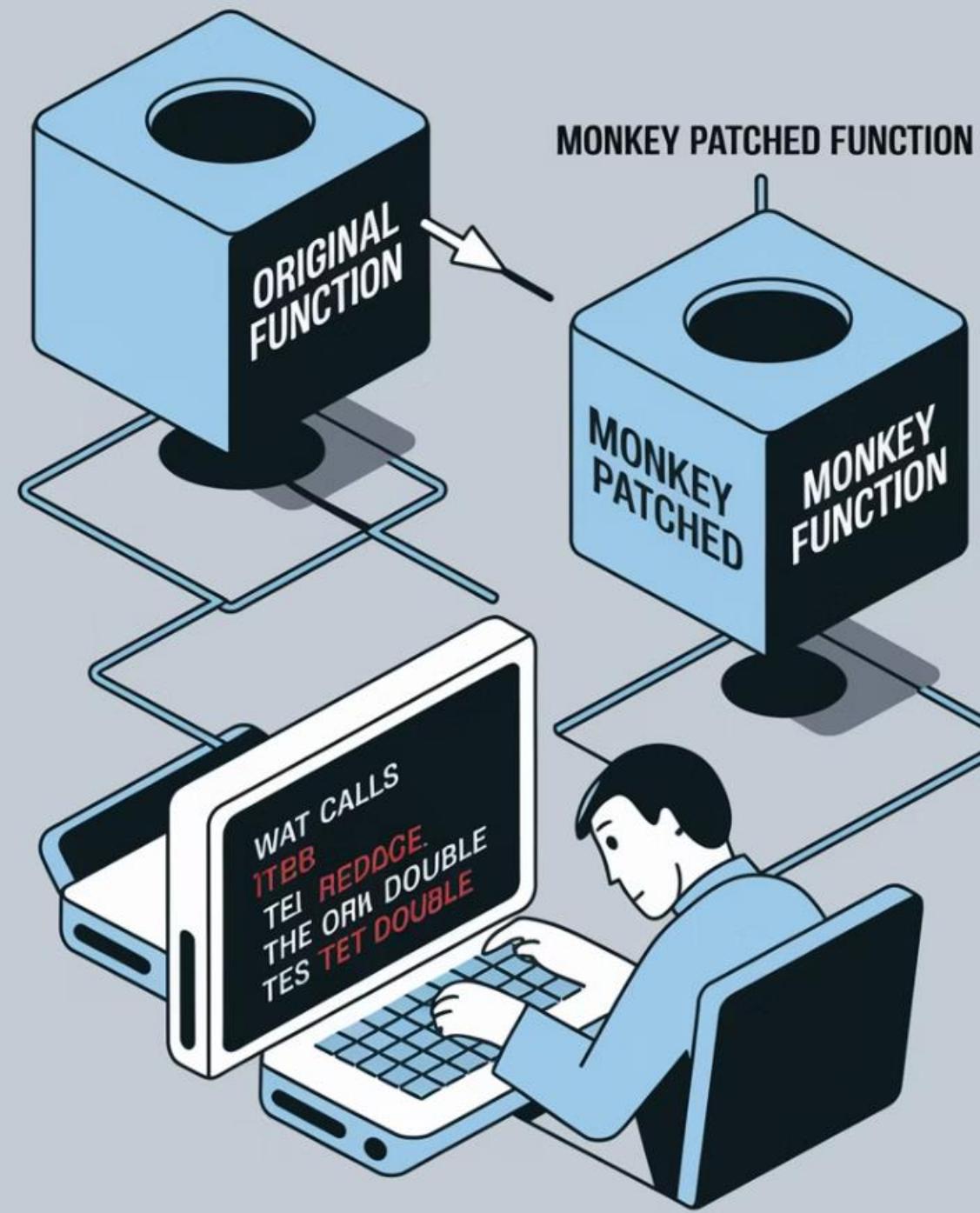
Parametrizácia testov - príklad

```
@pytest.mark.parametrize("a,b,vysledok", [  
    (2, 2, 4),  
    (10, -5, 5),  
    (0, 0, 0)  
])  
  
def test_sucet_param(a, b, vysledok):  
    assert sucet(a, b) == vysledok
```

Každý vstup vytvorí samostatný test. Veľmi užitočné pri testovaní hraničných hodnôt.

| TEST CASE | PARAMETERS | RESULT |
|--------------------|--------------------|--------------------|
| NIEME WIER. VARTB | NIEME WER. WIERB | PARAMETRS |
| -----* | -----* | -----* |
| PASSS ----- TAOU | PASSS ----- TASS9 | PASSS ----- TAO0 |
| -----* | -----* | -----* |
| PASSS ----- TAOU | PASSS ----- * | PASSS ----- T/181E |
| -----* | -----* | -----* |
| PASSS ----- TAOU | PASSS BISSED TASS9 | PASSS ----- TAOU |
| -----* | -----* | -----* |
| PASSS ----- * | PASSS ----- * | PASSS ----- * |
| -----* | -----* | -----* |
| PASSS ----- TAOU | PASSS ----- TASS9 | PASSS ----- TAO1E |
| -----* | -----* | -----* |
| PASSS ----- T/18ED | PASSS ----- * | PASSS ----- TAOU |
| -----* | -----* | -----* |

ORIGINAL FUNCTION



Mocker a monkeypatching

```
def ziskaj_cenu():
    # napr. API call na burzu
    return 100

def test_ziskaj_cenu(monkeypatch):
    monkeypatch.setattr("src.modul.ziskaj_cenu", lambda:
42)
    assert ziskaj_cenu() == 42
```

Používa sa na izoláciu kódu od externých závislostí.

Integrácia do CI (GitHub Actions)

```
name: Run Tests

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'
      - name: Install dependencies
        run: pip install -r requirements.txt
      - name: Run pytest
        run: pytest --cov=src
```



Časté chyby a nedostatky

```
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
19     výzva do funkcie test_ súčasne obsahuje výzvu erroneo:: :;
```

Chyba

Zabudnuté test_ v názve funkcie

Riešenie

pytest ignoruje

Vysoká závislosť na globálnych stavoch

Použiť fixture

Duplicita testov

Použiť @parametrize

Testy sú príliš všeobecné

Testovať jedno správanie = jeden test

Nepoužívanie with pytest.raises()

Výnimky nebudú zachytené

Odporúčania pre prax



Deklaratívny prístup

Tvoriť testy deklaračne a zrozumiteľne.



Kompletné pokrytie

Každá funkcia = aspoň jeden test.



Hraničné stavy

Pokrývať aj hraničné a chybové stavy.



Integrácia

Integrovať testy do každej fázy vývoja.



Meranie pokrytia

Merat' code coverage, ale nespoliehať sa naň slepo.

Úlohy pytest

1. Čo sú to Unit Testy?
2. Čo je to modul unittest?
3. Ako sa používajú unit testy?
4. Ako vyzerá základná štruktúra?
5. Čo sú to assert metódy?
6. Aké sú vhodné prípady použitia?
7. Kde nájdeme dokumentáciu?



Ako pracovať s Časom a Dátumami

AKREDITOVANÝ KURZ

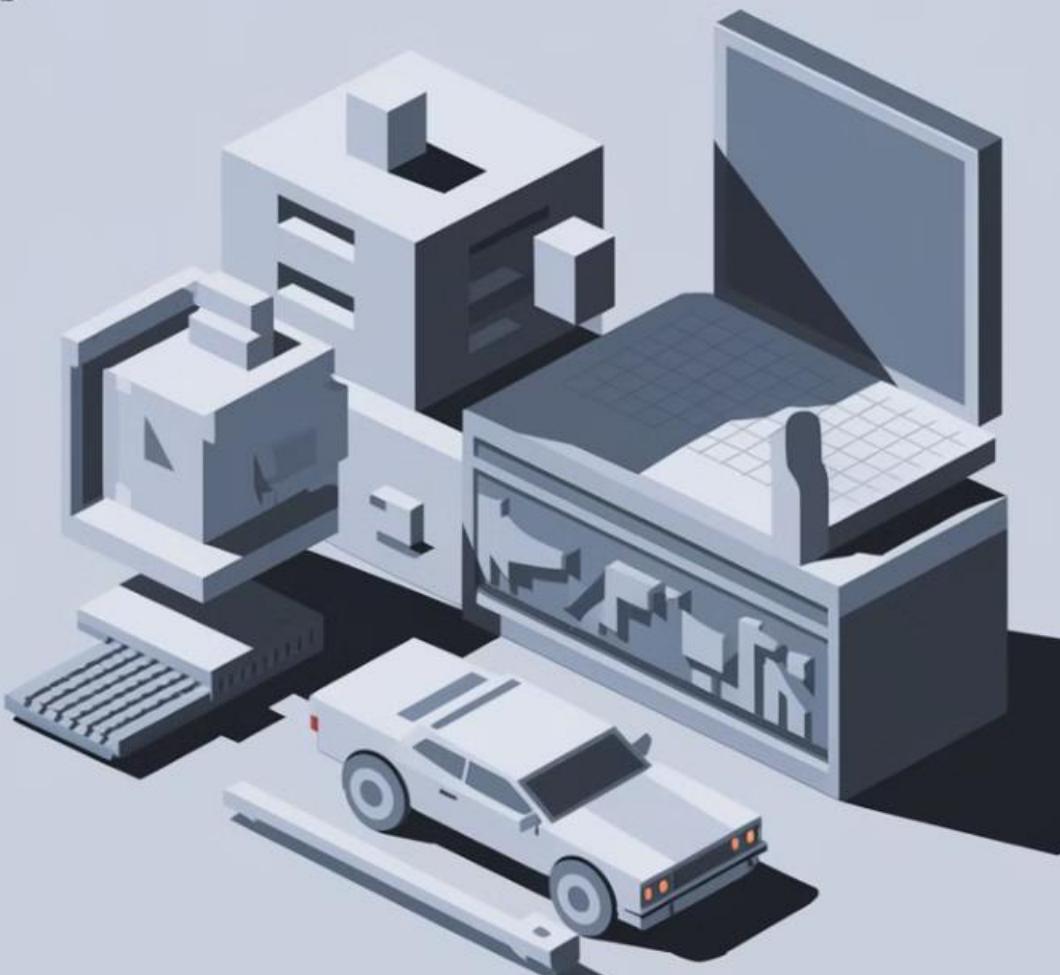




Čo sa naučíme?

1. Ako sa pracuje s dátumom?
2. Aké sú alternatívy?
3. Ako sa pracuje s náhodou?
4. Aké sú alternatívy?
5. Ako sa tieto moduly používajú?
6. Aké sú vhodné prípady použitia?
7. Kde nájdeme dokumentáciu?
8. Profit

```
1 urs([ary: [e
1 pynoe]
1 ionunrare([c]
1 nonrtrot([etons]) : ([cons, county, ([ee])
1 nrea[])
1 nenr [.
1 neoreaac]
1 netotoetoo([ers, 1-1][een
1 nootst([errsopoc[])
1 Cetoons-1-1
1 noutero([ettonts])
1 -([ee]), ([ceones ([Cns: ([etonsgn])
1
1s
```



Čo je to mockovanie?



Falošná verzia objektu

Nahrádza skutočnú implementáciu počas testovania.



Izolácia testovanej jednotky

Oddelenie funkcií a tried od ich závislostí.



Nahradenie externých prvkov

API, súbory, databázy dostanú predvídateľné správanie.

Kedy používať mockovanie

Nedostupné služby

Ked' nemáme prístup k reálnej službe ako API alebo databáza.

Pomalé alebo drahé testy

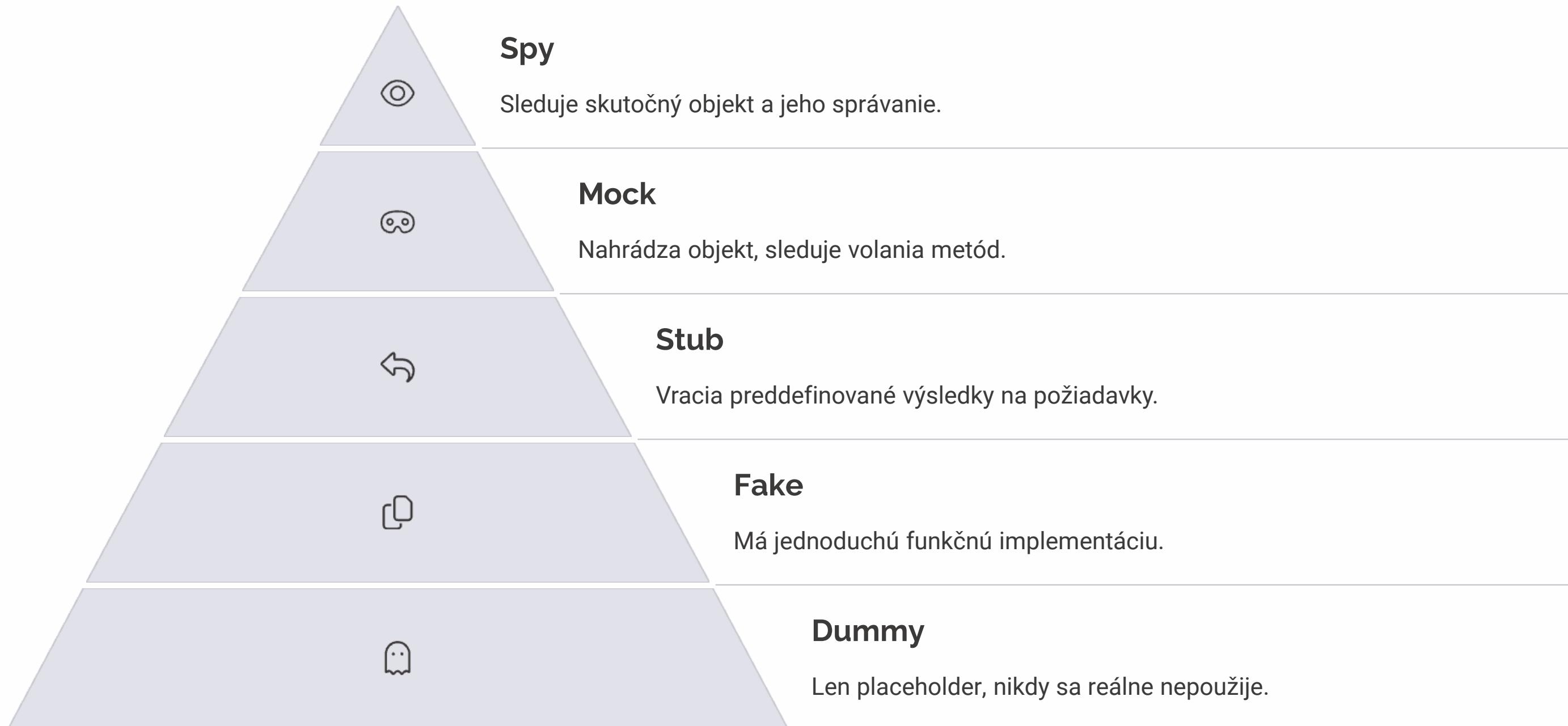
Ak je testovanie reálnej služby časovo náročné alebo nákladné.

Nestabilné prostredie

Pri potrebe konzistentných výsledkov v nestabilnom prostredí.



Typy testových dvojníkov



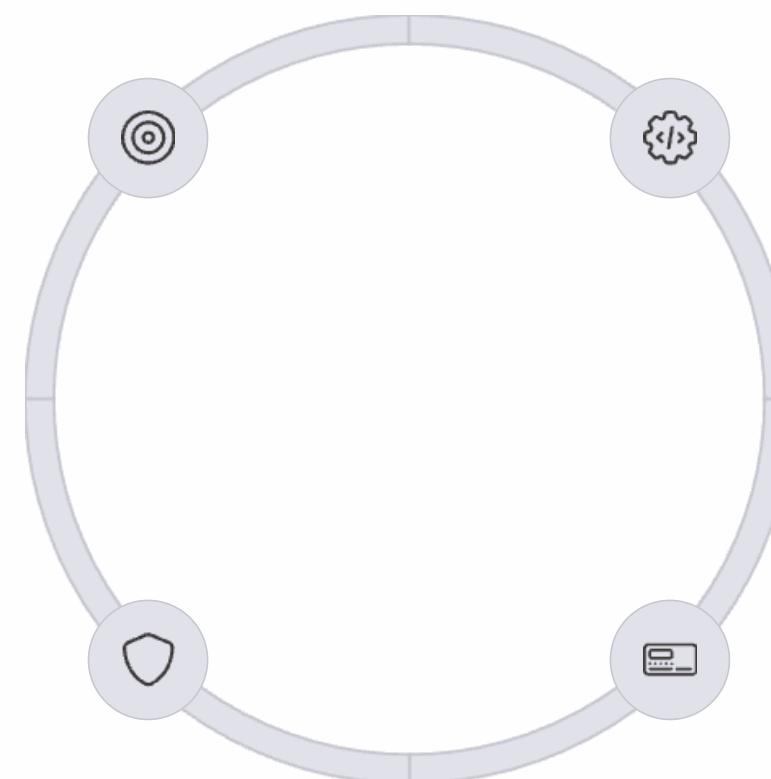
Cieľ mockovania

Izolácia kódu

Testujeme len tú časť, ktorú máme pod kontrolou.

Spoľahlivosť

Eliminácia nestabilných externých faktorov.



Predvídateľnosť

Nahradenie nepredvídateľných častí kontrolovaným správaním.

Rýchlosť

Zrýchlenie testov odstránením pomalých závislostí.

Prečo mockovať: Externé API

Problém

Testy zahŕňajú volania externého API.

- Spotreba dát
- Čakanie na odpoveď
- Závislosť od dostupnosti služby

Riešenie s mockovaním

Nahradenie API volania mockom.

- Okamžitá odpoveď
- Žiadne dátové limity
- Nezávislosť od externej služby

Prečo mockovať: Čas a náhoda



Časové funkcie

Testy s časom môžu byť nepredvídateľné alebo pomalé.



Náhodné hodnoty

Náhodné výsledky znemožňujú konzistentné testovanie.



Dátumy

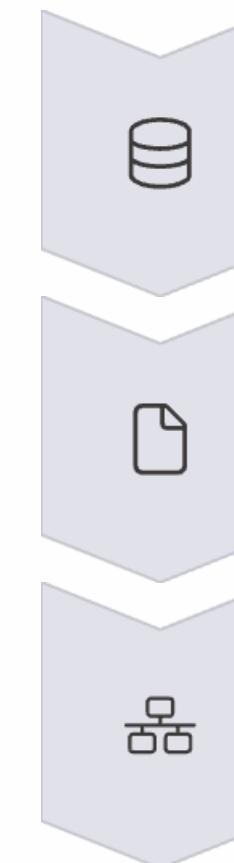
Funkcie závislé od dátumu potrebujú stabilné prostredie.

V tejto časti vysvetľujeme prečo je dôležité mockovať časové funkcie a náhodné hodnoty pri testovaní:

- Časové funkcie ako `sleep()` alebo `setTimeout()` môžu spôsobiť, že testy trvajú zbytočne dlho.
- Náhodné generátory (`Math.random`, `UUID`) produkujú nepredvídateľné výsledky, čo viedie k nestabilným testom.
- Funkcie závislé od aktuálneho dátumu a času (`new Date()`) môžu spôsobiť, že testy zlyhajú v rôzne dni alebo časti dňa.
- Mockovanie týchto funkcií zabezpečuje konzistentné a predvídateľné testovanie bez ohľadu na skutočný čas alebo náhodu.



Prečo mockovať: Databázy a súbory



Databázy

Testovanie bez migrácií a reálnych dát.

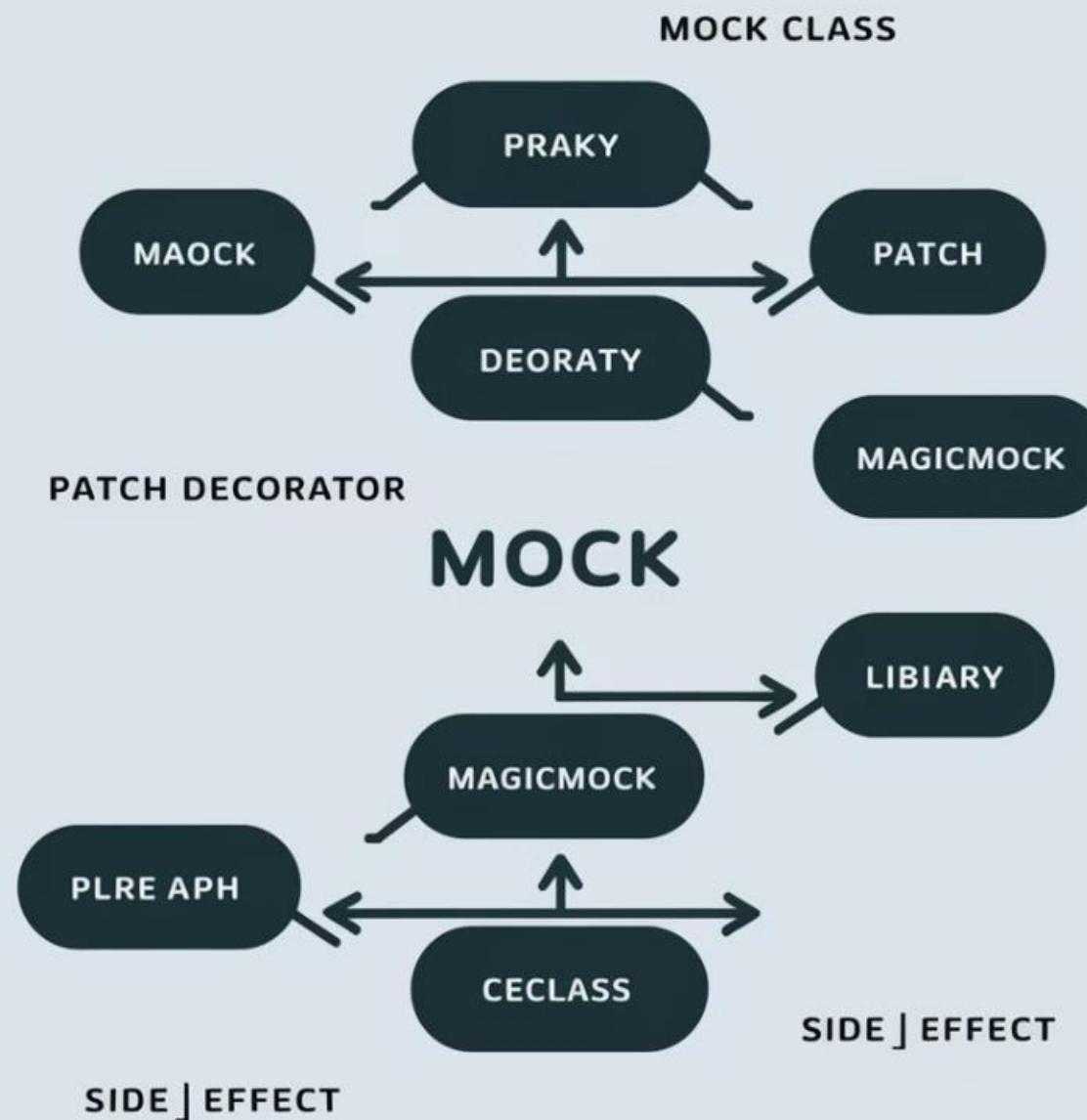
Súborový systém

Vyhnutie sa skutočným zápisom na disk.

Sieťové operácie

Možnosť testovať offline bez pripojenia.

Unittest.mock - kľúčové komponenty



Mock

Všeobecný náhradný objekt pre testovanie

MagicMock

Rozšírený Mock s podporou špeciálnych metód

patch

Nahradenie objektu v určenom namespace

patch.object

Nahradenie atribútu objektu alebo triedy

call, ANY

Nástroje na overovanie volaní a argumentov



Knižnica unittest.mock



Štandardná knižnica

Súčasť Pythonu od verzie 3.3.



Bohatá funkcionalita

Poskytuje Mock, MagicMock,
patch, call, PropertyMock.



Flexibilné použitie

Umožňuje nahradzať metódy,
funkcie, objekty, atribúty.

Základné použitie Mock

```
from unittest.mock import Mock

# vytvorenie falošnej funkcie
api = Mock()
api.get_data.return_value = {"id": 123, "name": "VITA"}

assert api.get_data() == {"id": 123, "name": "VITA"}
```

Importovanie

Importujeme Mock z unittest.mock.

Vytvorenie mocku

Vytvoríme inštanciu Mock pre simuláciu API.

Nastavenie správania

Definujeme očakávaný výsledok volania.

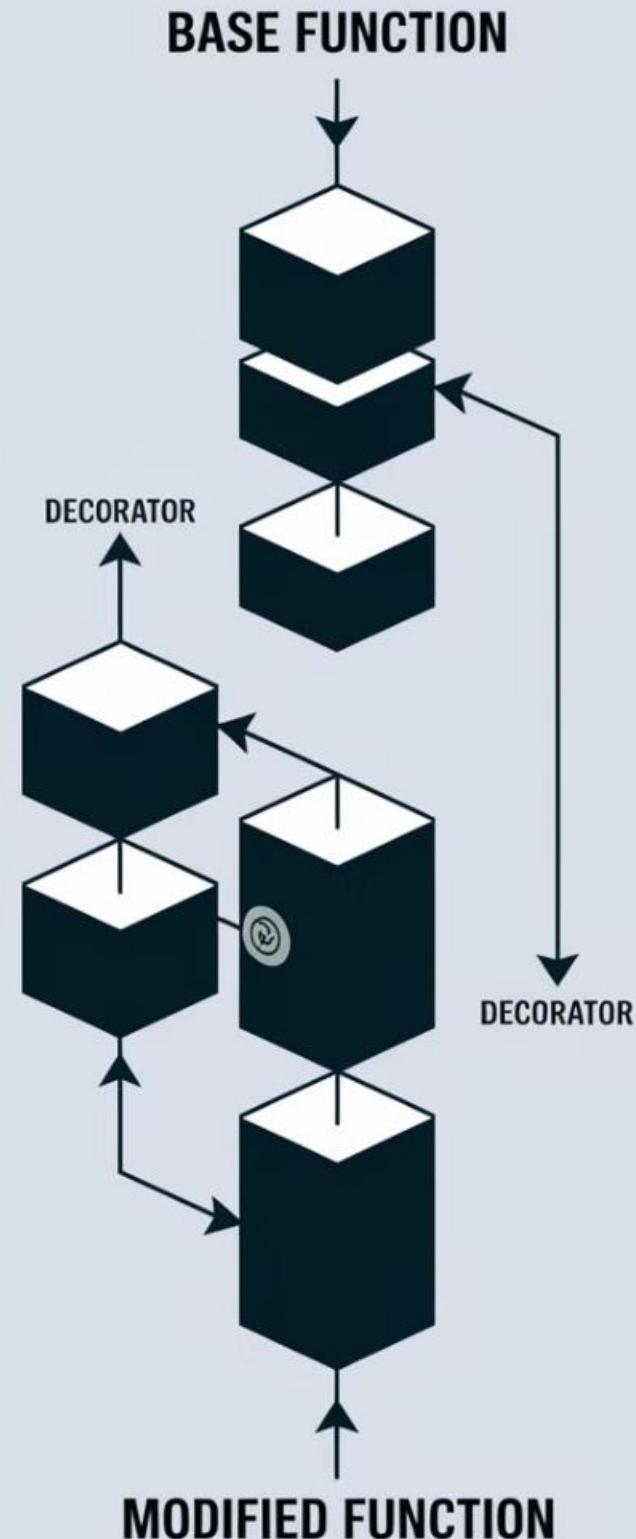
(fitroancikoation. ""
toiei)"P"
(ttooit.(iyooeibsth."S)"
(tteotheies. ""
titroooiitoahs."E)"
(fitrooaezio."|"
Pyth."|"
(fofroancipoatbeiet).ki
(fottoancibeeaih.(itjоthe.
(itroooaitrooah.(lhhd aiuiс.|.
socie.(i".
fieesiustе. ""
ih.(titazetbosies."|"
toooiirookd).!|
:|".

Použitie patch ako dekorátora

```
from unittest.mock import patch
from modul import nacitaj_data

@patch("modul.requests.get")
def test_nacitaj_data(mock_get):
    mock_get.return_value.json.return_value = {"id": 1}
    vysledok = nacitaj_data()
    assert vysledok["id"] == 1
```

Patch() dočasne nahradí cielovú funkciu objektom Mock. Používa sa pre testovanie HTTP volaní, databáz a súborov.



Použitie patch() ako kontextového manažéra

```
"wint:",  
avheleb.contentnt tocley.  
voglzo"goug",  
wh.contentesetatocie. - 92  
"wi h.2  
wtioo.contenant tocley.  
wh.contei.ea",  
wiht.contentesanote „...  
theleoootengssant tocte. - 92  
poleach-whnelly;  
wi h.2  
wh.contenteu",  
puleenano",  
uit"Celiaouy"
```

```
def test_api():  
    with patch("modul.requests.get") as mock_get:  
        mock_get.return_value.status_code = 200  
        mock_get.return_value.json.return_value = {"ok": True}  
        vysledok = nacitaj_data()  
        assert vysledok["ok"] is True
```



Vytvorenie kontextu

Otvorenie bloku with s patch().

Konfigurácia mocku

Nastavenie návratových hodnôt.

Testovanie funkcie

Volanie testovanej funkcie.

Overenie výsledku

Kontrola očakávaného výstupu.

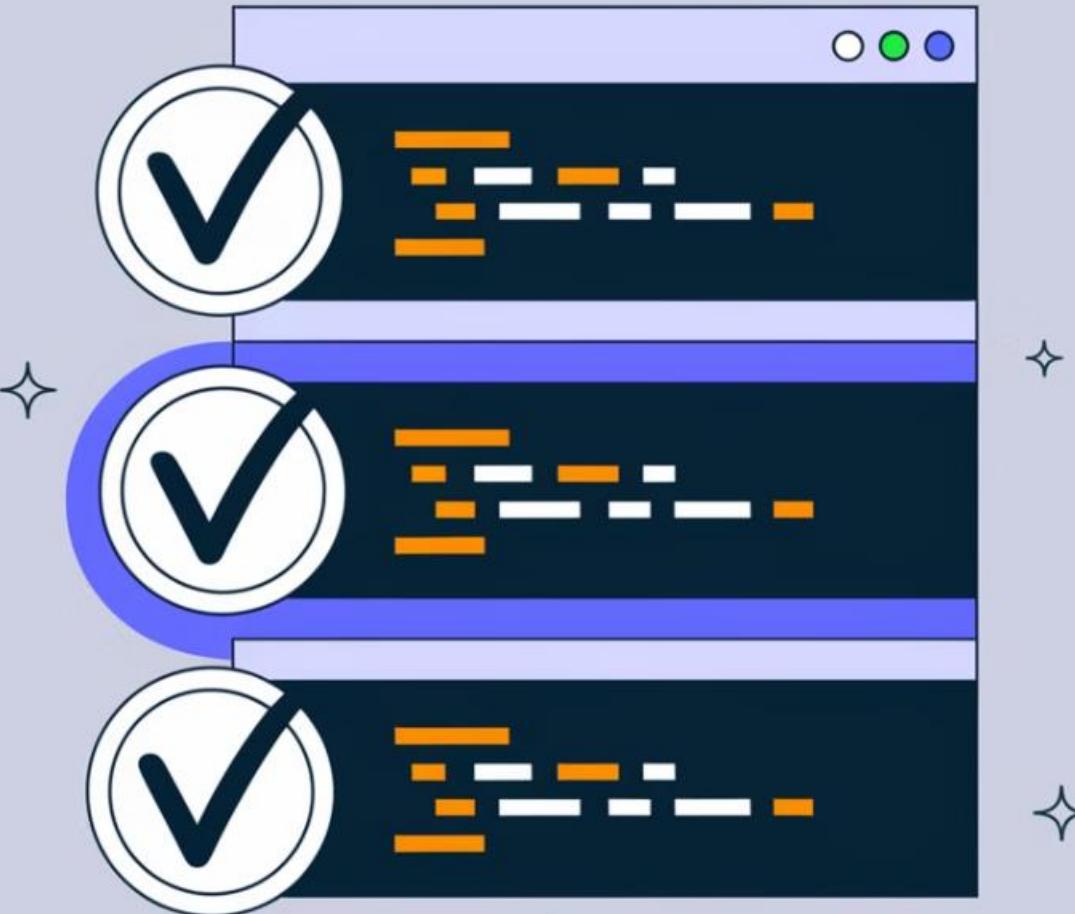
V tomto slajde vysvetľujeme použitie patch() ako kontextového manažéra pomocou konštrukcie with. Tento prístup je výhodný, keď chceme limitovať rozsah mockovania len na konkrétny blok kódu.

Kľúčové body:

- Kontext s with automaticky obnoví pôvodné funkcie po ukončení bloku
- Premenná mock_get obsahuje vytvorený mock objekt, ktorý môžeme konfigurovať
- Nastavenie return_value.json.return_value simuluje reťazené volania metód
- Tento prístup je čitateľnejší ako dekorátor pre komplexnejšie testy

Prípadné otázky poslucháčov: Aký je rozdiel medzi použitím patch ako dekorátora a kontextového manažéra? Odpoveď: Dekorátor sa aplikuje na celú testovaciu funkciu, zatiaľ čo kontextový manažér presne vymedzuje rozsah patchovania.

Overenie volania mocku



Testing Concept

```
mock.get_data.assert_called_once()  
mock.get_data.assert_called_with("parameter")
```

assert_called_once()

Overí, že metóda bola volaná presne jedenkrát.

assert_called_with(...)

Kontroluje, či bola metóda volaná so správnymi parametrami.

assert_not_called()

Overí, že metóda nebola vôbec volaná.

Pri testovaní s mockami nestačí iba nahradíť funkciu mockom, ale musíme aj overiť, či bol mock správne použitý. Na to slúžia metódy `assert_called_once()`, `assert_called_with()` a `assert_not_called()`.

Pomocou týchto metód môžeme kontrolovať:

- Či bola funkcia volaná správny počet krát
- Či bola volaná s očakávanými parametrami
- Prípadne či nebola volaná vôbec, ak to test vyžaduje

Je dôležité zdôrazniť, že overenie správneho použitia mockov je rovnako dôležité ako testovanie samotného výsledku.

Praktický príklad: testovanie HTTP API

Testovaný kód (api.py)

```
import requests

def ziskaj_pocasie(mesto):
    r =
    requests.get(f"https://api.pocasie.sk/{mesto}")
    return r.json()
```

Test (test_api.py)

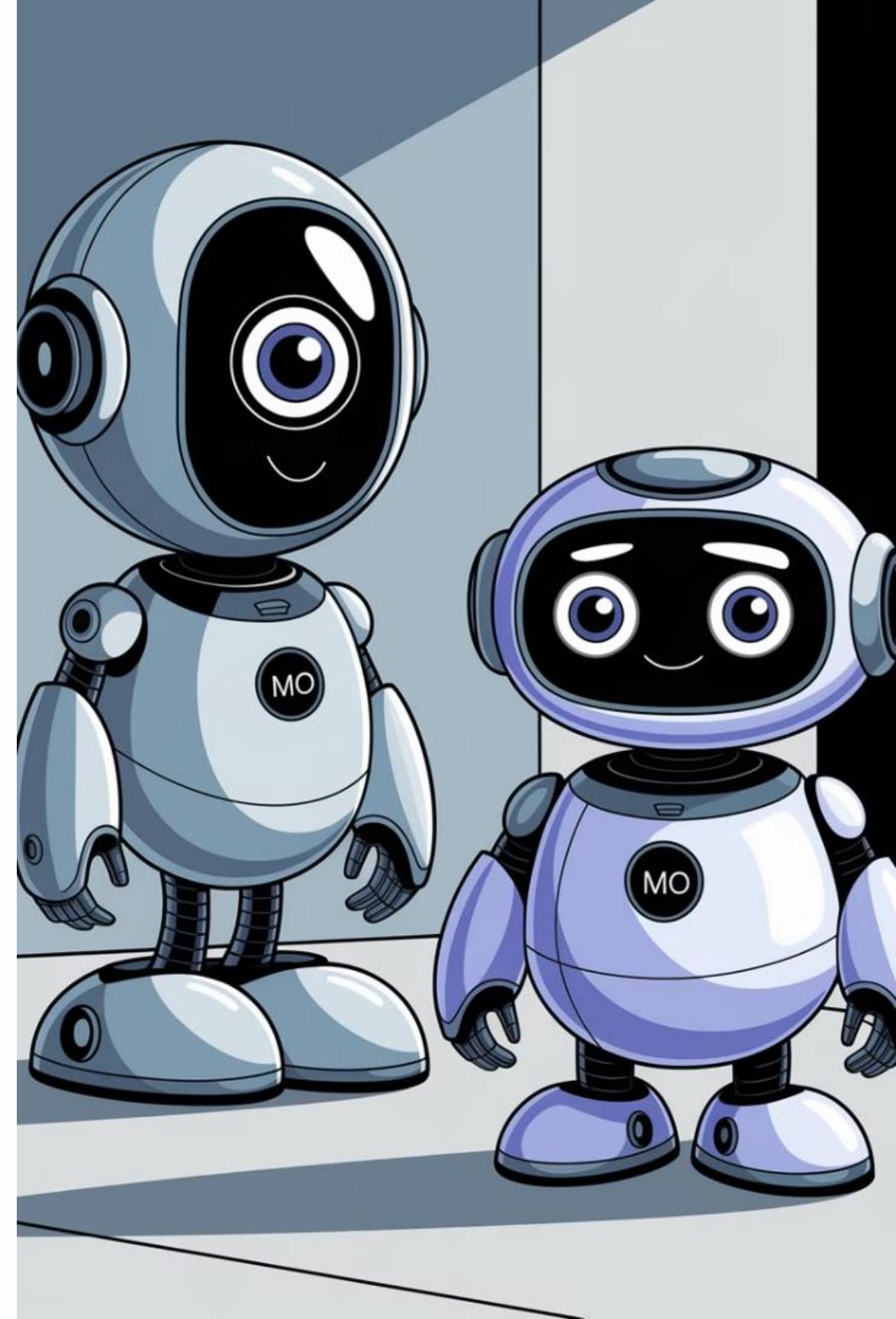
```
from unittest.mock import patch
from api import ziskaj_pocasie

@patch("api.requests.get")
def test_ziskaj_pocasie(mock_get):
    mock_get.return_value.json.return_value =
    {"teplota": 25}
    vysledok = ziskaj_pocasie("Bratislava")
    assert vysledok["teplota"] == 25
    mock_get.assert_called_once_with(
        "https://api.pocasie.sk/Bratislava")
```

Rozdiel medzi Mock a MagicMock

| Vlastnosť | Mock | MagicMock |
|---------------------------|------|-----------|
| Základná funkcia | ✓ | ✓ |
| Podpora špeciálnych metód | ✗ | ✓ |
| Vhodné pre OOP objekty | ✓ | ✓ |

MagicMock automaticky implementuje špeciálne metódy ako `__len__`, `__getitem__` a ďalšie, čo ho robí vhodnejším pre komplexné objekty.



Časté chyby pri mockovaní



Nesprávny namespace

Patchovanie zlej cesty (requests vs. modul.requests).



Chýbajúce return_value

Zabudnuté nastavenie pre vnorené metódy.



Externé závislosti

Závislosť testu od internetu alebo databázy.

Pri mockovaní sa vývojári často stretávajú s niekoľkými typickými problémami:

- Nesprávny namespace - Často sa stáva, že patchujeme nesprávnu cestu. Napríklad, ak kód používa 'requests' priamo, ale my patchujeme 'modul.requests', mock nebude fungovať. Je dôležité presne identifikovať, odkiaľ je modul importovaný v testovanom kóde.
- Chýbajúce return_value - Pri vytváraní mockov pre reťazené volania metód (napr. response.json()) je potrebné nastaviť return_value pre každú úroveň. Častá chyba je zabudnúť nastaviť návratovú hodnotu pre vnorené metódy, čo vedie k nečakaným AttributeError výnimkám.
- Externé závislosti - Písanie testov, ktoré závisia od externých zdrojov ako internet alebo databáza, je proti princípom unit testov. Tieto závislosti by mali byť vždy mockované, aby testy boli rýchle, spoľahlivé a nezávislé od externého prostredia.

Odporúčania pre prax



Selektívne mockovanie

Mockujte len to, čo neviete ovplyvniť.



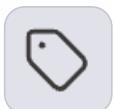
Konzistentné dáta

Mocky by mali vracať predvídateľné výsledky.



Zachovanie logiky

Nepoužívajte mocky na overovanie výpočtovej logiky.



Zrozumiteľné pomenovanie

Používajte jasné názvy pre mocky.



Ako funguje patch – namespace princíp

Zlaté pravidlo patchovania: Patchuj v tom module, kde sa objekt používa, nie kde je definovaný.

Nesprávne

```
@patch("requests.get") #
```

✗ patchuje originál

Tento prístup patchuje pôvodnú knižnicu, nie jej import v našom module.

Správne

```
@patch("modul.requests.get") #
```

✓ patchuje lokálny import

Tento prístup patchuje requests.get tak, ako je importovaný v našom module.

Patchovanie tried a metód

```
@patch("modul.ApiKlient.ziskaj_pocasie")
def test_pocasie(mock_metoda):
    mock_metoda.return_value = {"stav": "slnecno"}
    vysledok = aplikacia.pocasie()
    assert vysledok["stav"] == "slnecno"
```

Identifikácia cieľa

Určíme presnú cestu k metóde triedy.

Aplikácia patch dekorátora

Použijeme @patch s cestou k metóde.

Konfigurácia mocku

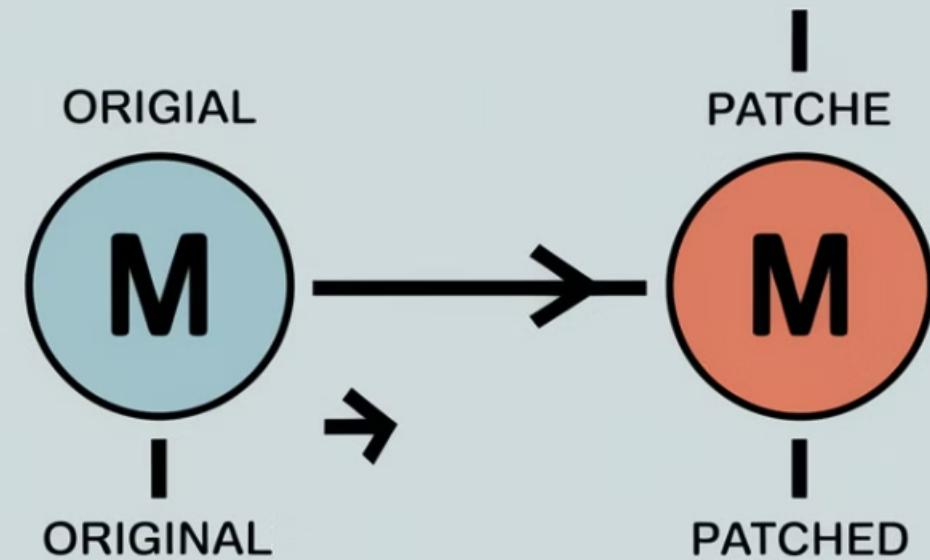
Nastavíme očakávaný výsledok metódy.

Pri patchovaní tried a metód je klúčové dodržiavať namespace princíp spomenutý v predchádzajúcim slajde.

Cesta k patchovanej metóde musí byť presne taká, ako je importovaná v testovanom module.

Ukážkový kód demonštruje kompletný proces - od patchovania metódy cez nastavenie návratovej hodnoty až po overenie, že testovaná funkcia používa mock správne.

Pripomeňte účastníkom, že patchovanie je možné aplikovať nielen pomocou dekorátorov, ale aj ako kontextový manažér s použitím 'with' bloku, čo je užitočné pri patchovaní len časti funkcie.



Patchovanie property

```
from unittest.mock import PropertyMock

@patch("modul.Objekt.cache", new_callable=PropertyMock)
def test_cache(mock_cache):
    mock_cache.return_value = {"session": "xyz"}
```

PropertyMock

Špeciálny typ mocku pre property atribúty.

new_callable parameter

Určuje typ mocku, ktorý sa má použiť.

Nastavenie hodnoty

Definuje, čo property vráti pri prístupe.

Pri testovaní tried s property atribútmi potrebujeme špeciálny prístup, keďže property nie sú bežné metódy ale deskriptory.

PropertyMock je špeciálna trieda v balíku unittest.mock, ktorá vie správne simulať Python property. Na rozdiel od štandardných mockov, PropertyMock sa správa ako property a umožňuje interceptovať prístup k property.

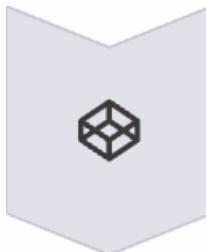
Pri patchovaní property musíme:

- Importovať PropertyMock z unittest.mock
- Použiť parameter new_callable=PropertyMock v patch dekorátore
- Nastaviť return_value namiesto priameho priradenia hodnoty

Tento postup je užitočný hlavne pri testovaní tried, ktoré používajú property na výpočty, cachovanie alebo prístup k externým zdrojom.

Vlastný mock objekt

```
class FakeApi:  
    def get_user(self, id):  
        return {"id": id, "meno": "Test"}  
  
def test_api(monkeypatch):  
    monkeypatch.setattr("modul.Api", lambda: FakeApi())
```



Vytvorenie triedy

Definujeme vlastnú triedu s potrebnými metódami.



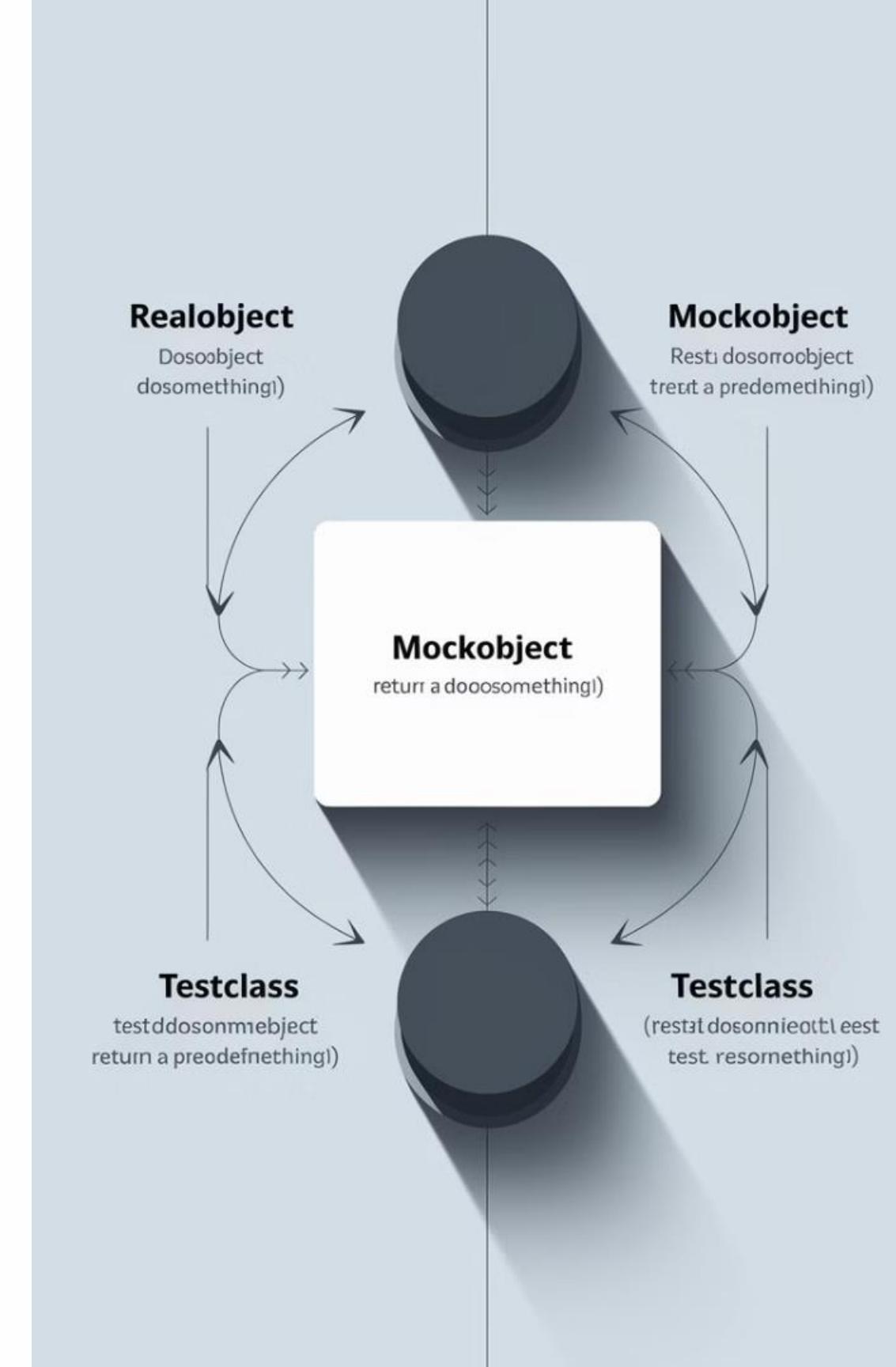
Nahradenie pôvodnej triedy

Použijeme monkeypatch na injekciu našej implementácie.



Testovanie s vlastným objektom

Kód bude používať našu implementáciu namiesto originálu.





Kontrola volania a argumentov

```
mock.get_data.assert_called_once()  
mock.get_data.assert_called_with(user_id=42)  
mock.get_data.call_count == 1  
mock.get_data.call_args == call(user_id=42)
```



assert_called_once()

Overí jedno volanie metódy.



assert_called_with()

Kontroluje argumenty posledného volania.



call_count

Počítadlo všetkých volaní metódy.



call_args

Detailed posledného volania metódy.

V tomto slajde vysvetľujeme, ako môžeme verifikovať volania mockovacích objektov a ich argumenty.

Pri testovaní s mockovacími objektmi je dôležité overiť nielen to, či boli metódy volané, ale aj akým spôsobom:

- **assert_called_once()** - pomocník, ktorý overí, že metóda bola volaná práve jedenkrát. Vyvolá výnimku, ak nebola volaná alebo bola volaná viackrát.
- **assert_called_with()** - overí, či posledné volanie metódy prebehlo s konkrétnymi argumentami. Veľmi užitočné pre kontrolu, či kód správne komunikuje s externými systémami.
- **call_count** - atribút, ktorý obsahuje počet všetkých volaní danej metódy. Užitočné, keď potrebujeme presne overiť počet volaní.
- **call_args** - atribút, ktorý poskytuje podrobné informácie o poslednom volaní vrátane pozičných a kľúčových argumentov.

Tieto techniky sú nevyhnutné pre robustné testovanie a pomáhajú zabezpečiť, že kód správne komunikuje s mockovacími objektmi.

Vzor – testovanie vrstvy use-case

Doména

```
class PouzivatelService:  
    def __init__(self, repo):  
        self.repo = repo  
  
    def prihlasenie(self, email):  
        return self.repo.najdi_podla_emailu(email)
```

Test

```
def test_prihlasenie():  
    mock_repo = Mock()  
    mock_repo.najdi_podla_emailu.return_value = {  
        "id": 1,  
        "email": "test@vita.sk"  
    }  
  
    service = PouzivatelService(mock_repo)  
    vysledok = service.prihlasenie("test@vita.sk")  
  
    assert vysledok["id"] == 1  
    mock_repo.najdi_podla_emailu.assert_called_once_with(  
        "test@vita.sk")
```

V tomto príklade vidíme typický vzor pre testovanie biznis logiky (use-case vrstvy). Tento prístup umožňuje testovať funkciaľitu bez závislosti na reálnej implementácii repozitára alebo databázy.

Kľúčové body tohto vzoru:

- Vytvoríme "mock" objekt pre repozitár, ktorý simuluje jeho správanie
- Definujeme očakávané návratové hodnoty pre metódy, ktoré sa budú volať
- Injektujeme mock do testovej služby cez konštruktor
- Overíme výsledok aj správne volanie metód repozitára

Tento vzor je užitočný, pretože nám umožňuje jednoducho testovať biznis logiku bez nutnosti pripájania sa k reálnej databáze, čo výrazne zrýchľuje testy a zjednoduší testovacie prostredie.

Refaktorovanie testov s fixture

```
@pytest.fixture  
def mock_requests_get(monkeypatch):  
    def fake_get(url):  
        class Odpoved:  
            def json(self):  
                return {"stav": "dobre"}  
        return Odpoved()  
  
    monkeypatch.setattr("modul.requests.get", fake_get)
```

Vytvorenie fixture

Definujeme znovupoužiteľnú fixture funkciu.

Konzistentné výsledky

Získame rovnaké správanie vo všetkých testoch.

Implementácia náhrady

Vytvoríme falošnú funkciu s požadovaným správaním.

Aplikácia v testoch

Používame fixture vo viacerých testoch.



Proti-patterny a chyby

Problém

Mockovanie bez overenia

Vysvetlenie

Nevieme, či bol mock vôbec volaný

Nadmerné mockovanie

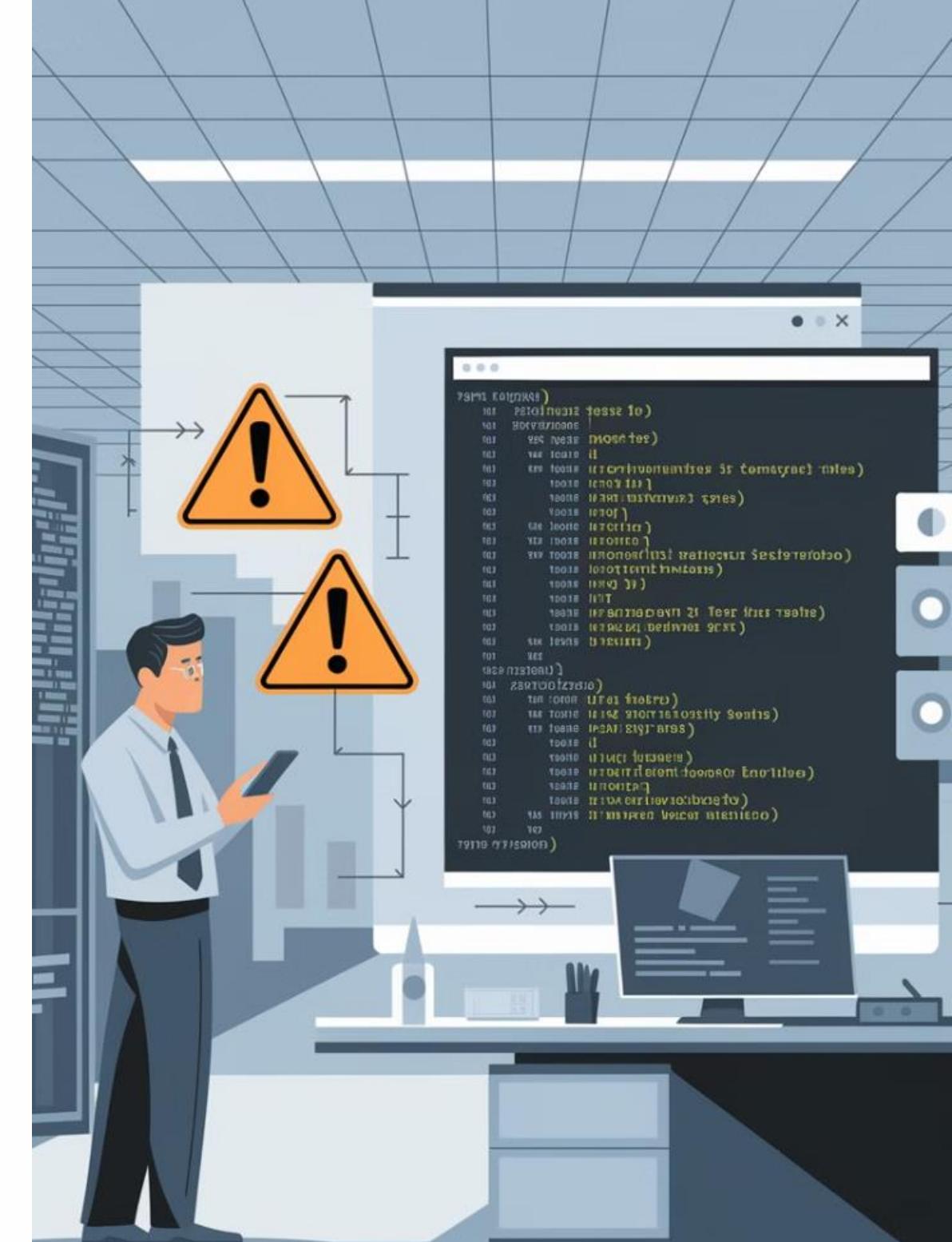
Testy strácajú hodnotu (mockujú všetko)

Závislosť na poradí volaní

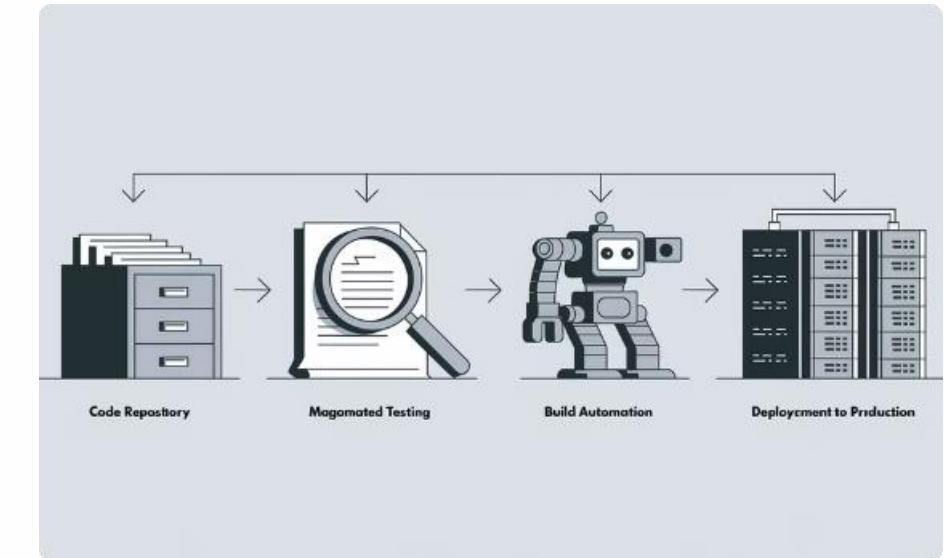
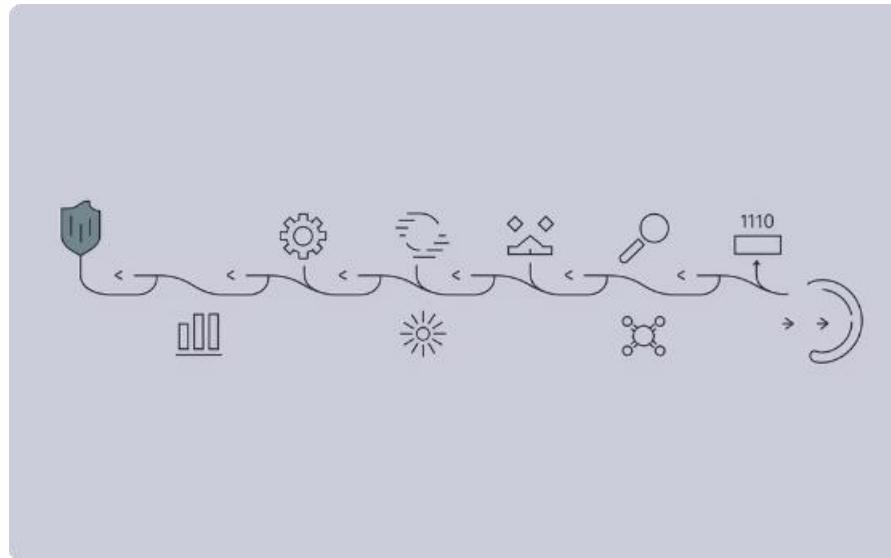
Testy sú krehké a závislé

Zlá namespace cesta pri patch

Patch sa nevzťahuje na cieľový objekt



Odporúčania do praxe



Externé závislosti

Mockujte len to, čo je externé, nestabilné alebo pomalé.

Vnútorná logika

Nepoužívajte mocky na vnútornú logiku, testujte skutočný kód.

Pri implementácii testovania so zameraním na mocky je dôležité dodržiavať tieto hlavné odporúčania:

- **Externé závislosti:** Mockujte iba systémy a služby, ktoré sú mimo vašej kontroly, ako napríklad API tretích strán, databázy, alebo služby s nestabilnym správaním. Tým zvýšite spoľahlivosť testov a znížite závislosť na externých faktoroch.
- **Vnútorná logika:** Vašu vlastnú biznis logiku by ste mali testovať s reálnymi implementáciami, nie s mockmi. Keď mockujete vlastný kód, znižujete hodnotu testov a môžete skryť chyby v implementácii.
- **CI/CD prístup:** V kontinuálnej integrácii je ideálne mať vrstvu rýchlych unit testov s mockmi pre rýchlu spätnú väzbu, ale tiež integračné testy bez mockov pre overenie skutočného správania systému.

Správne vyváženie mockovaných a nemockovaných testov viedie k robustnejšiemu testovaniu a kvalitnejšiemu kódu.

Dokumentácia a zdôvodnenie mockov

Dokumentujte účel

Vysvetlite, prečo daný mock existuje a čo nahradza.

Popíšte správanie

Uveďte, aké správanie mock simuluje a prečo.

Zdôvodnite rozhodnutia

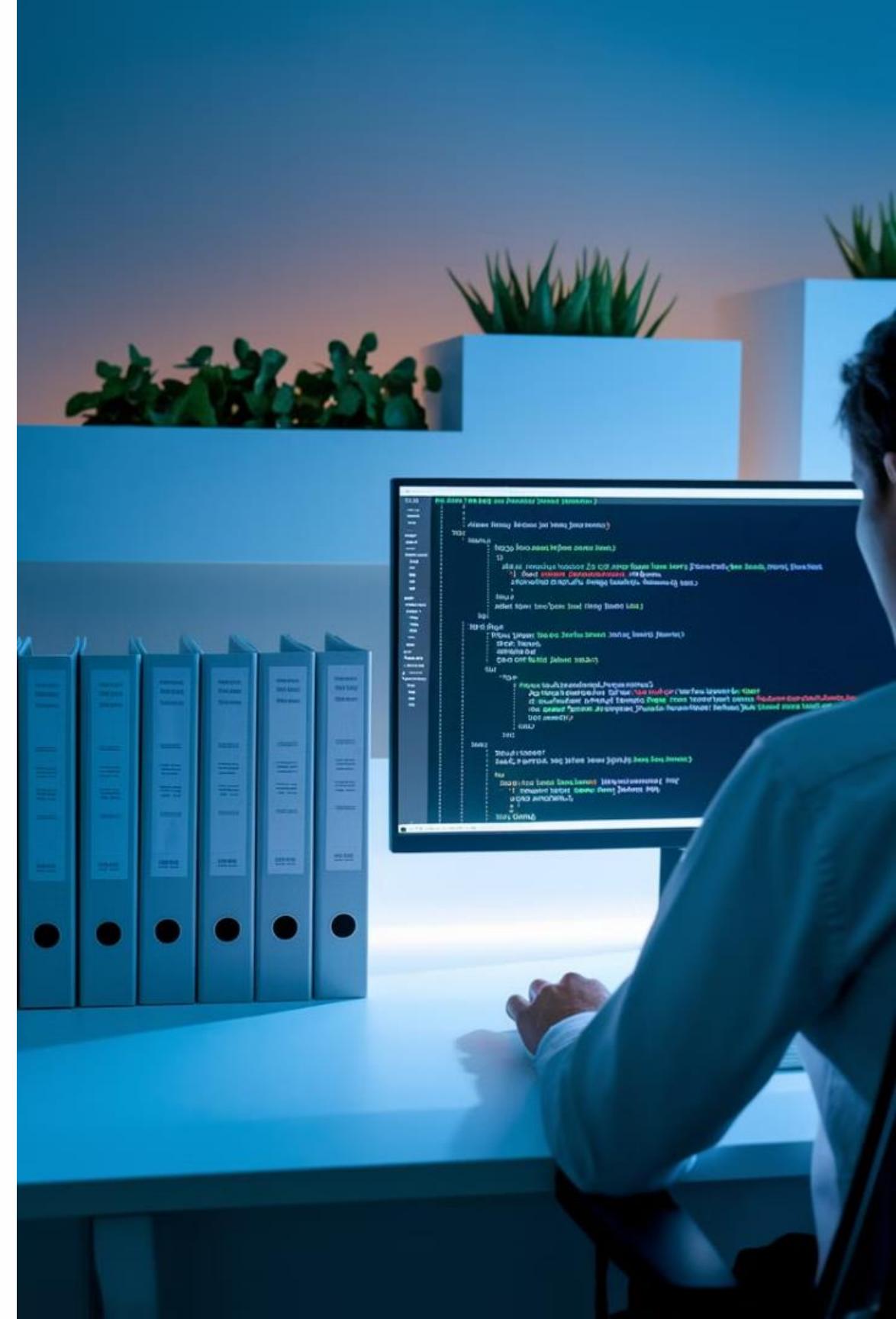
Vysvetlite, prečo ste sa rozhodli mockovať namiesto reálneho testovania.

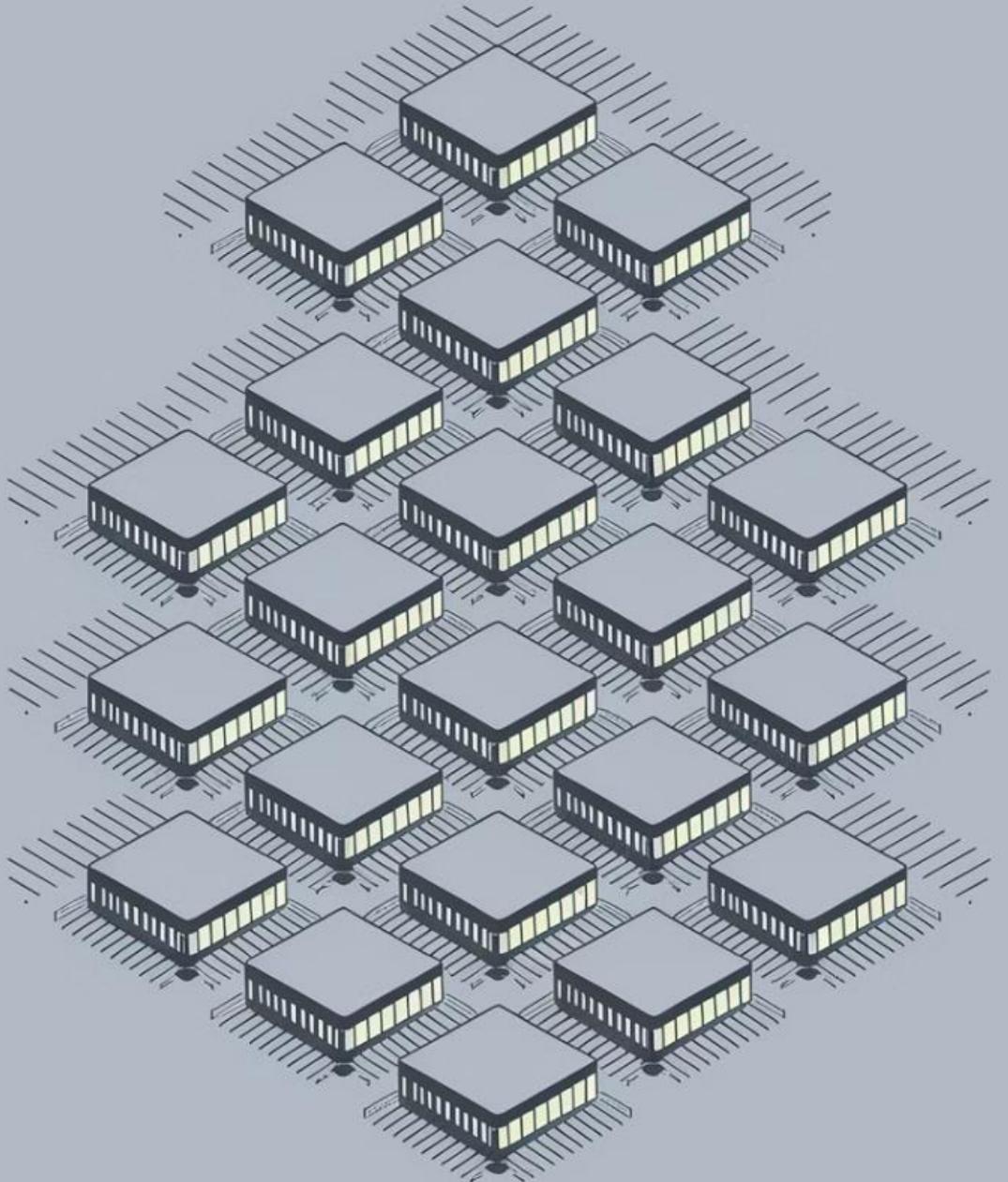
Pri práci s mockmi je dôležité udržiavať dôkladnú dokumentáciu, ktorá pomáha celému tímu pochopiť účel a fungovanie mockov v testoch.

Dokumentácia by mala obsahovať:

- Vysvetlenie, ktoré externé závislosti sú mockované a prečo
- Detailný popis simulovaného správania vrátane hraničných prípadov
- Zdôvodnenie, prečo je mockovanie vhodnejšie ako alternatívne prístupy (napríklad integračné testy)
- Príklady použitia mockov v testoch pre lepšie pochopenie

Dobrá dokumentácia mockov zjednoduší údržbu kódu a uľahčuje onboarding nových členov tímu.





Vlastné fake objekty pre komplexné správanie

```
class FakeDatabaza:  
    def __init__(self):  
        self.data = {}  
  
    def uloz(self, id, hodnota):  
        self.data[id] = hodnota  
  
    def nacitaj(self, id):  
        return self.data.get(id)  
  
    def vymaz(self, id):  
        if id in self.data:  
            del self.data[id]
```

Pre komplexné správanie je často lepšie vytvoriť vlastný fake objekt namiesto komplikovaného nastavovania mockov.

Príklad: Mockovanie času

Pôvodný kód

```
import time

def get_current_timestamp():
    return int(time.time())

def is_expired(timestamp, ttl=3600):
    current = get_current_timestamp()
    return (current - timestamp) > ttl
```

Test s mockom

```
@patch('modul.get_current_timestamp')
def test_is_expired(mock_time):
    # Nastavíme aktuálny čas
    mock_time.return_value = 1000

    # Test pre nevypršaný čas
    assert not is_expired(500, ttl=600)

    # Test pre vypršaný čas
    assert is_expired(300, ttl=600)
```

Príklad: Mockovanie súborového systému

```
@patch('builtins.open', new_callable=mock_open, read_data='test data')
def test_read_file(mock_file):
    with open('dummy_path.txt', 'r') as f:
        data = f.read()

    assert data == 'test data'
    mock_file.assert_called_once_with('dummy_path.txt', 'r')
```

Patchovanie open

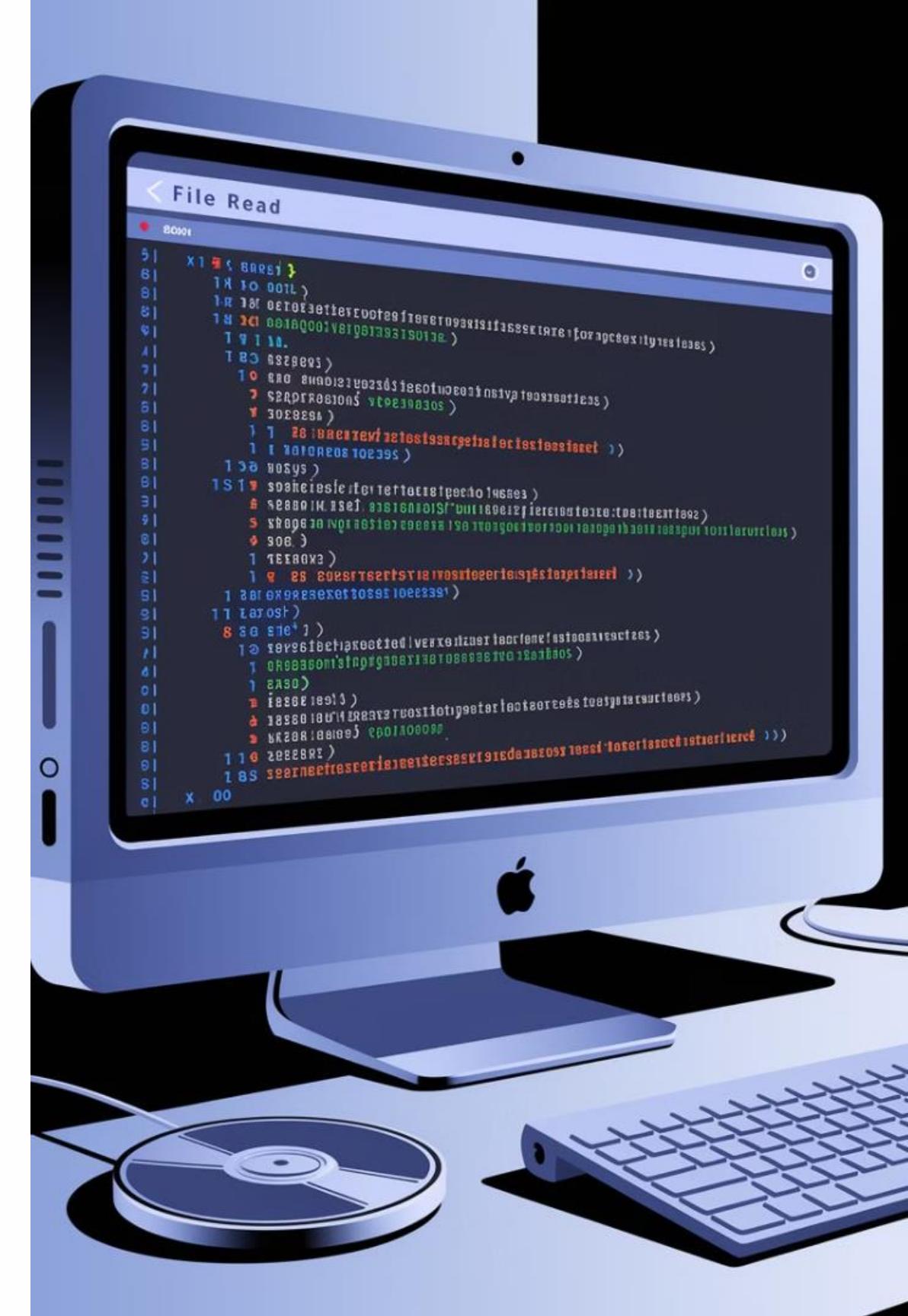
Nahradíme vstavaný open pomocou mock_open.

Simulácia obsahu

Definujeme, aké dátá má súbor obsahovať.

Testovanie čítania

Overíme, že kód správne číta dátá zo súboru.



Príklad: Mockovanie HTTP odpovede



```
@patch('requests.get')
def test_api_client(mock_get):
    mock_response = Mock()
    mock_response.status_code = 200
    mock_response.json.return_value = {'data': 'test'}
    mock_get.return_value = mock_response

    result = api_client.fetch_data('endpoint')
    assert result == {'data': 'test'}
```

Príklad: Mockovanie databázy

Vytvorenie mock objektu

Nahradenie databázového pripojenia mockom.

Testovanie logiky

Overenie správneho spracovania dát z databázy.

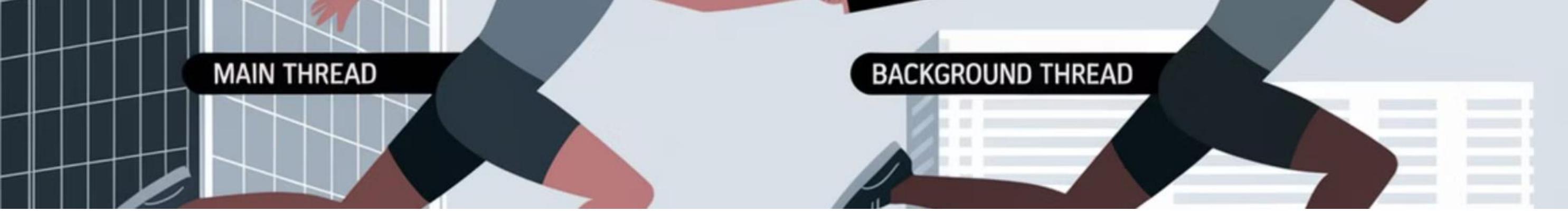
Simulácia dát

Definovanie očakávaných výsledkov dotazov.

Overenie dotazov

Kontrola správnych parametrov v SQL dotazoch.





Príklad: Mockovanie asynchronných funkcií

```
import asyncio
from unittest.mock import patch, AsyncMock

@patch('modul.async_api_call', new_callable=AsyncMock)
async def test_async_function(mock_api):
    mock_api.return_value = {'status': 'success'}

    result = await process_data()

    assert result == {'status': 'success', 'processed': True}
    mock_api.assert_called_once()
```



AsyncMock

Špeciálny mock pre asynchronné funkcie.

Asynchronny test

Test musí byť definovaný ako asynchronná funkcia.

Await volanie

Použitie await pre asynchronné operácie.

Zhrnutie kurzu



5

Typov testových dvojníkov

Dummy, Fake, Stub, Mock a Spy.

3

Spôsoby použitia patch

Dekorátor, kontextový manažér a priame volanie.

4

Hlavné dôvody mockovania

Externé API, čas, databázy a súborový systém.

Mockovanie je kľúčová technika pre efektívne testovanie softvéru. Umožňuje izolovať testovaný kód a simulať externé závislosti. Správne použitie mockov viedie k rýchlejším, spoľahlivejším a lepšie udržiavateľným testom.

Úlohy Mock

1. Čo sú to Unit Testy?
2. Čo je to modul unittest?
3. Ako sa používajú unit testy?
4. Ako vyzerá základná štruktúra?
5. Čo sú to assert metódy?
6. Aké sú vhodné prípady použitia?
7. Kde nájdeme dokumentáciu?

