

Pyladies meetup, Budapest, 2023. november 6.

Karbantartható és tiszta kód írása

Útmutató
a Python Zen és a Tiszta Kód
elveinek alkalmazásához

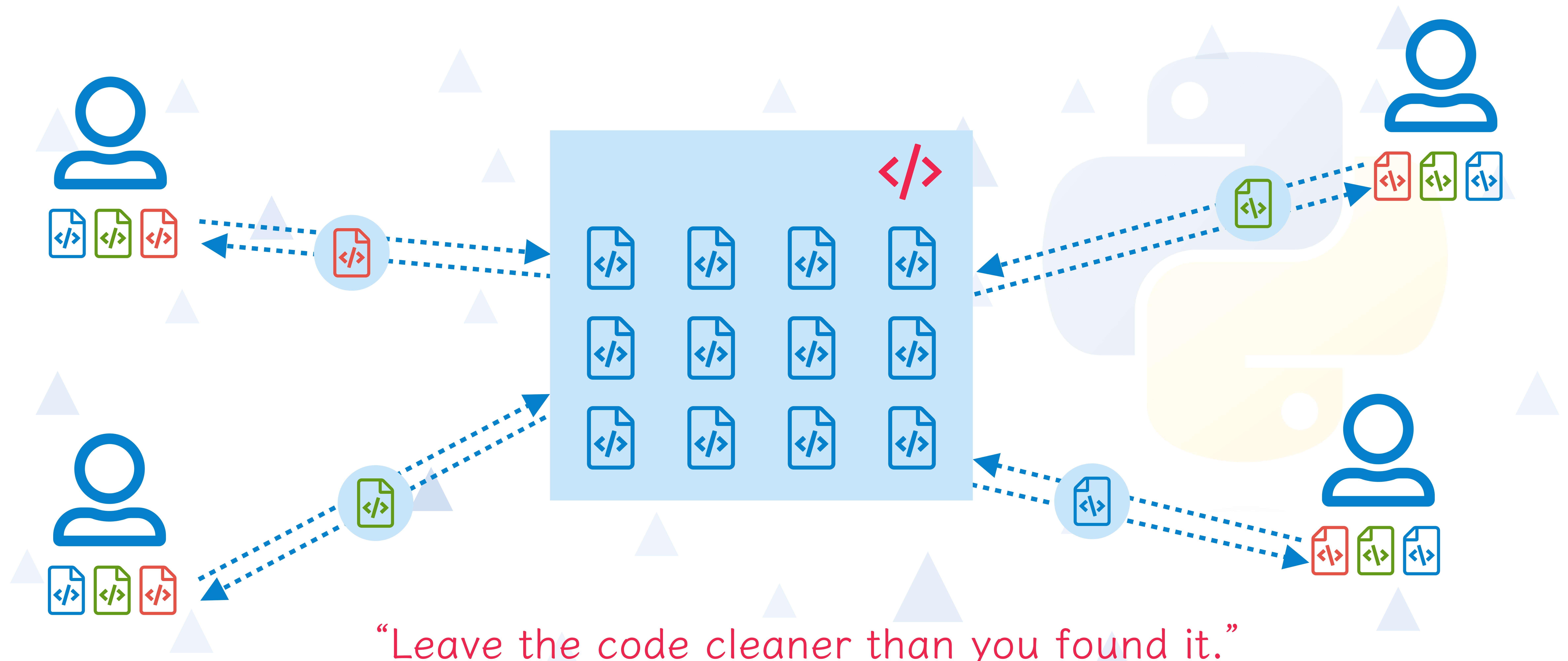
Fülöp Péter, peteristvan.fulop@hcl.com



- Irányelvek a **“Pythonic” stílus**ban történő kód írásához
- A “Pythonic” stílus alkalmazása az **olvashatóság és egyszerűség** érdekében
- A helyes **elnevezési konvenciók** fontossága
- Moduláris és **újrafelhasználható** kód írása
- A **kódstruktúra optimalizálása** a jobb szervezettségért
- Hatékony **dokumentációs technikák**



Miért van szükség konvenciókra kód írásakor?



“Leave the code cleaner than you found it.”

Every time that we make a change in the code base, make an improvement in the code.

The Zen of Python



A Python Zenje

A Python Zenje

- A **Python Zenje** irányelvek, axiómák, aranyszabályok gyűjteménye a “Pythonic” stílusban törénő programozáshoz, kód írásához.
- Ezeket az irányelveket először **Tim Peters, a Python egyik alapító közreműködője** mutatta be **1999**-ben. Első alkalommal Python programozók levelezőlistáján (comp.lang.python/python-list@python.org) lett publikálva. A “The Way of Python” című szálon lírai formában tették közzé.
- **2004**-ben az irányelveket beépítették a **PEP20** (Python Enhancement Proposals - Python Fejlesztési Javaslatok) [<https://peps.python.org/pep-0020/>]
- Ma a Python Zenjét a **nyelv alapvető filozófiájának tekintik**, és ez szolgál a Python programozás legjobb gyakorlatainak alapkövéül.

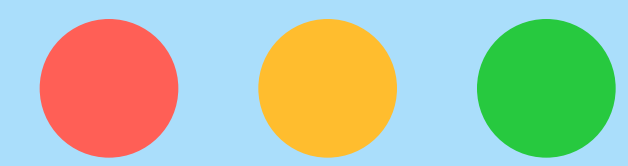
A Python Zenje

A Python Zenje



PYTHON

```
import this # 2001: Barry Warsaw added The Zen of Python into the Python language
```



OUTPUT

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.

Unless explicitly silenced.
In the face of ambiguity, refuse
the temptation to guess.
There should be one-- and
preferably only one --obvious way
to do it.
Although that way may not be
obvious at first unless you're
Dutch.
Now is better than never.



A Python nyelv fejlesztésének folyamata

PEPs (Python Enhancement Proposals)

- A **PEP** egy olyan **dokumentum**, amely a Python programozási nyelv új funkciójának, fejlesztésének vagy módosításának **javaslatát** részletezi.
- A **PEP folyamat** a fő mechanizmus a Pythonba történő jelentős változtatások javaslatára. Keretet biztosít a közösség számára a javaslatok megvitatására és értékelésére. Bárki benyújthat egy PEP-et, és **a Python közösség áttekinti és megvitatja a javaslatot**, mielőtt döntés születik arról, hogy elfogadják vagy elutasítják azt.
- A PEP-ek széles körű témákat ölelhetnek fel, az **új szintaxis és nyelvi elemek** bevezetésétől kezdve a könyvtármodulokon át a kódolási konvenciók szabványosításáig.
- A PEP dokumentumok listája a <https://peps.python.org/pep-0000/> cím alatt érhető el.

Példa pár jelentősebb elfogadott fejlesztési javaslatra

- **PEP 8: Útmutató Python kód stílusához, formázásához.** Útmutatást nyújt a konzisztens, olvasható és könnyen karbantartható Python kód írásához. Olyan témákat foglal magában, mint a névkonvenciók és a kód elrendezése.
- **PEP 20: A Python Zenje.** Ez a PEP vázolja a Python nyelv és közösségének irányadó elveit, beleértve az egyszerűséget, olvashatóságot és a gyakorlatiasság fontosságát.
- **PEP 257: Docstring konvenciók.** Ez a PEP útmutatást ad a dokumentáció(docstringek) írásához, amelyek leírják a Python modulok, függvények, osztályok és metódusok célját, használatát és viselkedését.
- **PEP 484: Típusjelölések (Type Hints).** Ez a PEP bevezet egy szintaxist az opcionális típusjelölések megadásához a Python kódban.

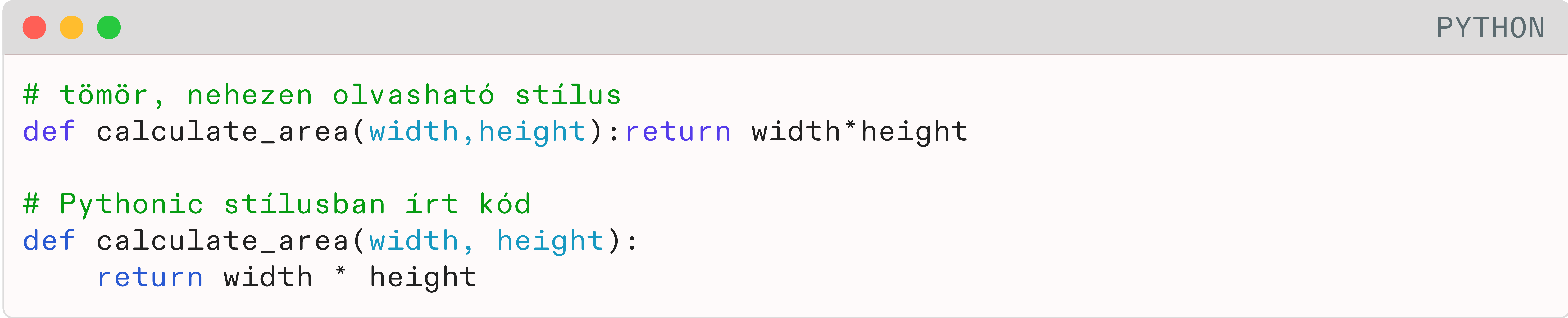
Olvashatóság, egyszerűség

“Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ... [Therefore,] *making it easy to read makes it easier to write.*” — Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

“Readability counts.” — The Zen of Python

A kód olvashatóságát segítő konvenciók

- 4 szóközből álló behúzás használata a kódblokkok jelzésére (kerüljük a TAB használatát).
- Kódsorok röviden tartása (ideális esetben legfeljebb 80 karakter hosszan).
- Szóköz használata az operátorok körül és a vesszők után a jobb olvashatóság érdekében.



```
# tömör, nehezen olvasható stílus
def calculate_area(width,height):return width*height

# Pythonic stílusban írt kód
def calculate_area(width, height):
    return width * height
```


A kód olvashatóságát segítő konvenciók

→ **Sortöréshez** zárójelek `()`, `[]`, `{}` javasolt a **Python implicit sorfolytatásának** kihasználásával

PYTHON

```
list_of_positive_integers = [  
    1, 2, 3,  
    4, 5, 6  
]  
  
long_and_verbose_variable_name = {  
    'key1': 'lorem ipsum dolor amit',  
    'key2': 'long string goes here',  
    'key3': 'many characters goes here' ,  
}
```

Változók, függvények, osztályok, konstansok elnevezése

- snake_case használata a **függvényekhez és változókhoz**
- nagybetűs CamelCase az **osztályokhoz**
- **metódusnevek:** kisbetűs szó vagy szavak, aláhúzásokkal elválasztva
- rövid, kisbetűs szó vagy szavak a **modulok** nevéhez. Használjunk aláhúzásokat a szavak elválasztásához.
- kizárólag nagybetű a **konstansok** elnevezéséhez
- **csomagnevek:** rövid, kisbetűs szó vagy szavak. Ne használjunk aláhúzásokat az elválasztáshoz

Példa elnevezési konvenciókat nem követő kódra

PYTHON

```
# PEP8 elnevezési konvenciókat nem követő kód
```

```
Max_Retries = 10
```

```
def GetAverageOfSquares(Numbers):
```

```
    return sum([number * number for number in Numbers]) / len(Numbers)
```

```
class custom_user:
```

```
    def __init__(self, FullName, EmailAddress, CodedPassword):
```

```
        self.FullName=FullName
```

```
        self.EmailAddress=EmailAddress
```

Példa elnevezési konvenciókat követő kódra

PYTHON

```
# PEP8 elnevezési konvenciókat követő kód
```

```
MAX_RETRIES = 10
```

```
def get_average_of_squares(numbers):  
    return sum([number * number for number in numbers]) / len(numbers)
```

```
class CustomUser:  
    def __init__(self, full_name, email_address, coded_password):  
        self.full_name = full_name  
        self.email_address = email_address  
        self.coded_password = coded_password
```


Olvashatóság növelése az “import”-ok rendszerezése

PYTHON

```
# olvashatóságot nem támogató elrendezés
```

```
import numpy as np
from utils import helper_function
import os
from my_module import my_function
import sys
import pandas as pd
```

PYTHON

```
# olvashatóságot támogató elrendezés
```

```
import os
import sys

import numpy as np
import pandas as pd

from my_module import my_function
from utils import helper_function
```

Elnevezési konvenciók - hogy válasszunk jó megnevezéseket?

Változók, függvények, osztályok, konstansok elnevezése

- A használt elnevezés **kommunikálja jól a szándékot** ne legyen félrevezető, bizonytalan jelentésű
- **Legyen kiejthető** (az esetleges szóbeli kommunikáció miatt), tartsa be a helyesírási szabályokat
- Változók, osztályok elnevezésére **használjunk főnevet**, ugyanakkor kerüljük a túl általános fogalmazásokat: pl. Manager, Info, Data stb. A név ne utaljon annak típusára, zavaró
- Metódus és függvények elnevezésére **használjunk igét** vagy igei alakot
- Ugyanazon koncepcióra **egységesen** ugyanazon megnevezést használjuk (pl. *retrieve*, *fetch*, *get*). Több, eltérő koncepcióra ne használjunk ugyanazon/hasonló megnevezést, megtévesztő
- A megnevezések **kapjon elegendő kontextust** kapjon (number: *phone_number* or *house_number*?) de többet ne a szükségesnél

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

— Martin Fowler

“Linter”-ek, automatikus formázók (autoformatters)

→ **LINTER:** A linterek elemzik programjainkat és jelzik a benne található “hibákat” (eltéréseket adott konvencióktól). Javaslatokat adnak a hibák kijavítására.

pylint (gyakorlati példával)

Flake8

pycodestyle

→ **AUTOFORMATTER:** Automatikus formázók újrastrukturálják a Python kódunkat, hogy megfeleljenek a PEP 8 stílus útmutatónak. Az automatikus formázók általában nem olyan szigorúak, mint a linterek. Egy megbízható linter szól nekünk, ha a kódunk túl bonyolult, de egy automatikus formázó nem fogja ezt megtenni.

autopep8

Black

Moduláris és újrafelhasználható kód írása



Modularitás, újrafelhasználhatóság

A modularitás és az újrafelhasználhatóság a tiszta kód alapvető elvei. Amikor a kódod moduláris és újrafelhasználható, könnyebb karbantartani, hibát keresni és bővíteni.

→ **Kiss, fókuszált függvények írása**, amelyek egy feladatot jól végeznek el. **A függvényeknek egyetlen felelősségük kell, hogy legyen**, és ne legyenek néhány tucat kódsornál hosszabbak.

```
PYTHON

# több felelősséggel rendelkező függvény

def process_user_data(user_data):
    # validate user data
    # save user data to the database
    # send confirmation mail
```

```
PYTHON

# egyetlen felelősséggel rendelkező függvények

def validate_user_data(user_data):
    pass

def save_user_data_to_db(user_data):
    pass

def send_confirmation_mail(user_data):
    pass
```


Modularitás, újrafelhasználhatóság

- Korlátozzuk a függvény argumentumainak számát. Ideális esetben **egy függvény ne vegyen több mint három vagy négy argumentumot**.
- **Szervezzük az összetartozó függvényeket osztályokba**. Ez elősegíti az enkapszulációt (egységbe zárást) és megkönnyíti a kód kezelését és újrafelhasználását.
- **Kerüljük a "mindenható objektumokat"** (God objects) vagy osztályokat, amelyek túl sok mindent próbálnak megcsinálni. Bontsuk le őket kisebb, fókuszáltabb komponensekre. (**SRP alkalmazása**)
- **Használjuk a Python beépített könyvtárait**: időt takaríthatunk meg, csökkentheti a kód bonyolultságát, és elkerülhetjük a kerék újra feltalálását. Amikor egy beépített könyvtár nem felel meg az igényeknek, akkor vizsgáljuk meg további csomagokat a Python Csomag Indexről (PyPI).



A kódstruktúra optimalizálása a jobb szervezetségért

Egy jól strukturált kódbázis könnyebben áttekinthető,
megérthető és karbantartható.

Modulok, csomagok, kivételkezelés

- A kód **modulokba** és **csomagokba** szervezése
- A megfelelő **kivételkezelés** megvalósítása
 - `try` és `except` blokkok használata a kivételek elkapásához és kezeléséhez
 - kizárólag csak azokat a specifikus kivételeket kapjuk el, amelyeket tudunk kezelni vagy helyreállítani tud, és hagyjuk, hogy a többi továbbterjedjen
 - lássuk el a kivételeket elegendő információval ahhoz, hogy segítsen a probléma diagnosztizálásában.
 - kerüljük az üres `except`-ek használatát, mivel azok nem szándékosan el tudnak kapni kivételeket, és megnehezíthetik a hibakeresést.

Megfelelő kivételkezelés

PYTHON

```
# kizárólag olyan kivételeket fogunk ki, amit kezelni tudunk
def get_file_content(file_path):
    try:
        with open(file_path, "r") as file:
            return file.read()
    except FileNotFoundError:
        print(f"File not found: {file_path}")
    except IOError:
        print(f"An error occurred while reading the file: {file_path}")
```


Világos és tömör dokumentáció írása

A dokumentáció a tiszta kód létfontosságú része.

A jól dokumentált kód könnyebben érthető, karbantartható és bővíthető.

Docstringek

- Pythonban a docstringek jelentik a függvények, osztályok és modulok dokumentálásának **szabványos módját**
- **Függvények/metódusok esetén** egy jól megírt docstringnek tartalmaznia kell egy rövid leírást a függvény által ellátott feladatról, a várt bemenetokről és kimenetekről.

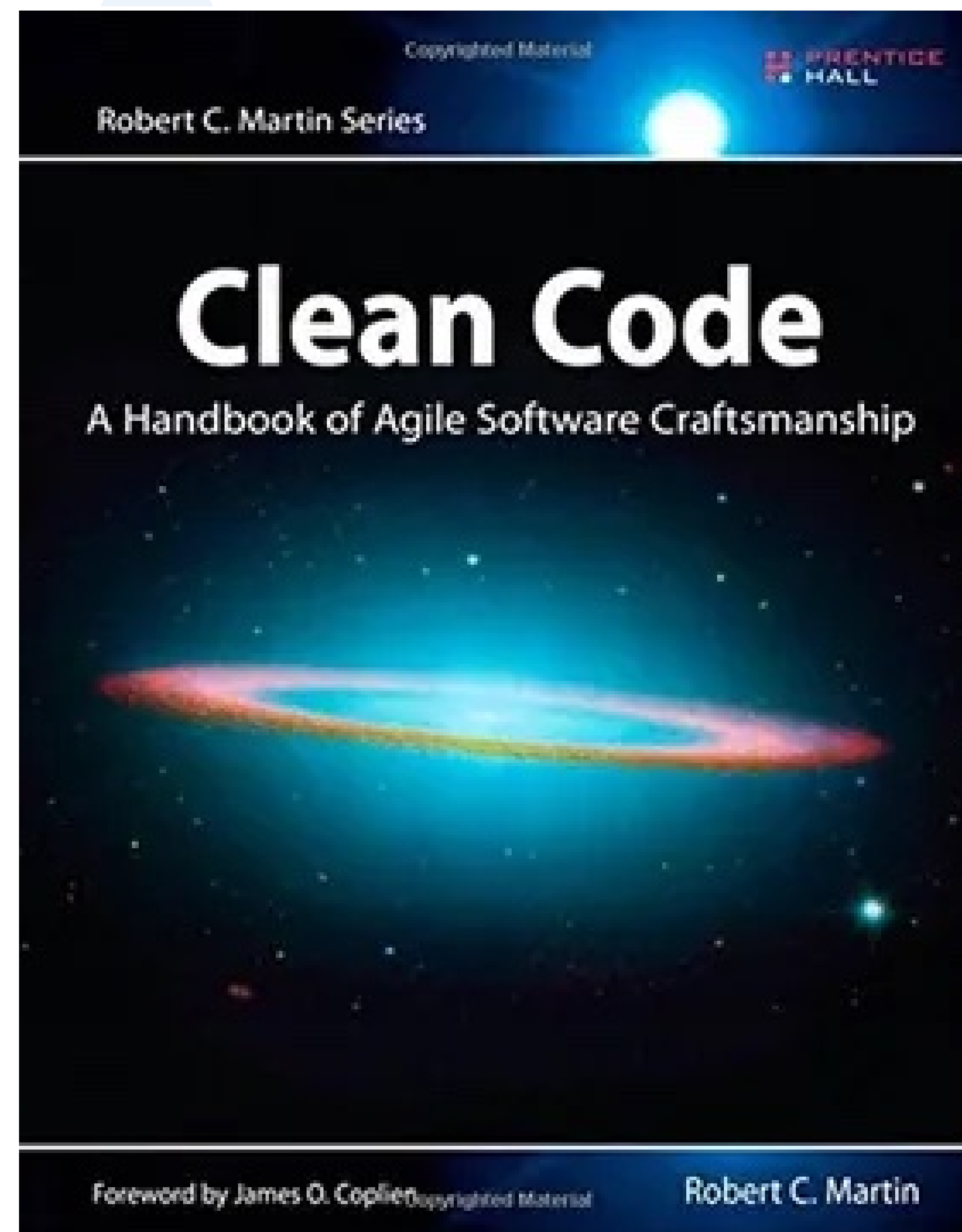
PYTHON

```
def calculate_age(birthdate: datetime.date, today: Optional[datetime.date] = None) → int:
    """
    Calculate a person's age based on their birthdate.Args:
        birthdate (datetime.date): The person's birthdate.
        today (Optional[datetime.date]): The reference date to calculate age from.
        Defaults to None (uses the current date).
    Returns:
        int: The person's age in years.
    """
    # Implementation...
```


Docstringek

- Kövessünk egy **konzisztens formátumot**, mint például a Sphinx, NumPy vagy Google docstring stílusok.
- **Kezdjünk egy egy-soros összefoglalóval**, szükség esetén ezt követheti egy részletesebb magyarázat.
- **Ismertessük minden bemeneti argumentum célját és használatát**, valamint a függvény visszatérési értéket.
- **Tartsuk a docstringeket naprakészen** a kód változtatásakor.

"Clean Code: A Handbook of Agile Software Craftsmanship" című könyv is kiemeli



A kód, amit írunk,
nem csak egy gép számára készül,
hanem emberek számára is ...