CLEAN CODE PART 1

برنامه سازی پیشرفته، دکتر زهرا قربانعلی
نگارش: سامیار قاسمی، عسل محمدجعفری
دانشگاه صنعتی امیرکبیر
دانشکده ریاضی و علوم کامپیوتر
بهار ۱۴۰۴

کلین کد (Clean Code) چیست؟

کلین کد (Clean Code) مجموعه‌ای از قواعد، شیوه‌ها و اصول طراحی است که به بهبود خوانایی، نگهداری و توسعه‌پذیری کد کمک می‌کند. در دنیای واقعی، برنامه‌نویسان بیشتر کد یکدیگر را می‌خوانند تا اینکه کد جدید بنویسند؛ بنابراین نوشتن کدهایی که هم برای کامپیوتر و هم برای انسان قابل فهم باشد، از اهمیت ویژه‌ای برخوردار است. همانطور که مارتین فاولر گفته است:

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

این جمله تاکید می‌کند که مسئولیت ما به عنوان توسعه‌دهندگان تنها در ایجاد عملکرد صحیح کد نیست، بلکه در ایجاد کدی است که به آسانی توسط تیم‌های دیگر و حتی خودمان در آینده قابل نگهداری و بهبود باشد.

اصول نام‌گذاری

نام‌گذاری مناسب اولین قدم برای نوشتن کد تمیز است. انتخاب نام‌های معنادار، خوانایی کد را افزایش داده و به دیگران کمک می‌کند تا منظور و نقش هر متغیر یا تابع را به راحتی درک کنند.

سبک‌های رایج نام‌گذاری

- **snake_case:**

تمامی حروف به صورت کوچک نوشته شده و کلمات با علامت زیر خط (_) از هم جدا می‌شوند.

- **CamelCase:**

کلمات به‌هم متصل شده و حرف اول کلمه اول کوچک و اولین حرف کلمات بعدی بزرگ نوشته می‌شود.

- **PascalCase:**

در این سبک، تمامی کلمات به‌هم متصل شده و حرف اول هر کلمه به صورت بزرگ نوشته می‌شود.

متغیرها

انتخاب نام مناسب برای متغیرها به وضوح و مفهوم کد کمک فراوانی می‌کند. نکاتی که در انتخاب نام برای متغیرها باید مد نظر قرار گیرد عبارتند از:

استفاده از اسامی توصیفی (Noun):

به جای استفاده از حروف یا مخفف‌های نامفهوم، از اسامی کامل و توضیح‌دهنده استفاده کنید.

به عنوان مثال:

```
active_user_amount = 55
```

به جای:

```
au = 55  # number of active users
```

قابل تلفظ بودن:

نام‌هایی انتخاب کنید که نه تنها معنی داشته باشند بلکه به راحتی قابل تلفظ و به خاطر سپردن باشند.

به عنوان نمونه:

```
generation_datetime = datetime.strptime('01/01/70 00:00:00',
'%m/%d/%y %H:%M:%S')
```

به جای:

```
genyyyymmddhhmmss = datetime.strptime('01/01/70 00:00:00',
'%m/%d/%y %H:%M:%S')
```

اجتناب از مخفف‌های غیر استاندارد:

استفاده از اسامی کامل و واضح به جلوگیری از ابهام کمک می‌کند.

به جای:

```
fna = 'Tom'
cre_tmstp = 19319874
```

ترجیحاً از:

```
first_name = 'Tom'
creation_timestamp = 19319874
```

استفاده نمایید.

یکدست بودن نام‌ها:

برای مفاهیم مشابه از یک واژه یکنواخت استفاده کنید تا از سردرگمی ناشی از استفاده از مترادف‌ها جلوگیری شود.

به عنوان مثال:

```
client_first_name = 'spongebob'
client_last_name = 'squarepants'
```

به جای:

```
client_first_name = 'spongebob'
customer_last_name = 'squarepants'
```

عدم استفاده از اعداد جادویی:

اعداد بدون توضیح که به طور مستقیم در کد ظاهر می‌شوند، خوانایی را کاهش می‌دهند.

به جای:

```
chosen_player = random.randint(1, 16)
```

از:

```
number_of_players = 16
chosen_player = random.randint(1, number_of_players)
```

استفاده کنید.

ذکر نوع داده در نام (در صورت لزوم):

افزودن پسوندهایی مانند _list یا _dict می‌تواند نوع داده متغیر را مشخص کند، مثلاً:

```
numbers_list = [1, 6, 9]
words_dict = {'a': 'apple', 'b': 'banana', 'c': 'cherry'}
```

توابع

توابع اصلی‌ترین اجزای برنامه هستند که وظایف خاصی را انجام می‌دهند. طراحی صحیح توابع به سادگی و مقیاس‌پذیری کد کمک می‌کند.

نکات مهم در طراحی توابع

استفاده از فعل برای نام توابع:

نام تابع باید عملکرد آن را به وضوح بیان کند، مانند get_name() یا calculate_total().

سادگی و کوتاهی:

هر تابع باید یک وظیفه مشخص داشته باشد و بیش از حد پیچیده نشود.

به عنوان نمونه، به جای:

```
def get_and_save_data():
    pass
```

بهتر است دو تابع جداگانه تعریف شود:

```
def get_data():
    pass

def save_data():
    pass
```

یکپارچگی در نام‌گذاری:

برای عملکردهای مشابه از یک الگوی نام‌گذاری ثابت استفاده کنید.

به عنوان مثال:

```
def get_name():
    pass

def get_age():
    pass
```

به جای استفاده از نام‌های متفاوت مانند get_name() و fetch_age().

اجتناب از پارامترهای flag:

استفاده از پارامترهای بولی برای تغییر مسیر اجرای یک تابع ممکن است کد را پیچیده و غیرقابل پیش‌بینی کند.

به جای:

```
def transform(text: str, upper_case: bool) -> str:
    if upper_case:
        return text.upper()
    else:
        return text.lower()
```

بهتر است دو تابع مجزا تعریف کنید:

```
def upper_case(text: str) -> str:
    return text.upper()

def lower_case(text: str) -> str:
    return text.lower()
```

کامنت‌گذاری

کامنت‌ها باید تنها در صورت نیاز برای توضیح «چرا» کد نوشته شده‌اند و نه «چطور» کار می‌کند. اصول کامنت‌گذاری صحیح عبارتند از:

کد خودتوضیح:

اگر کد به گونه‌ای نوشته شده که معنای خود را منتقل می‌کند، نیازی به کامنت اضافی نیست.

توضیح چرایی وجود کد:

کامنت‌ها باید توضیح دهند که چرا یک قطعه کد به شکلی نوشته شده و چه مشکلی را حل می‌کند.

حذف کدهای کامنت‌شده:

کدهایی که دیگر استفاده نمی‌شوند (مثلاً کدهای آزمایشی یا دیباگ) باید پیش از انتشار پاک شوند تا از ایجاد ابهام جلوگیری شود.

استاندارد PEP 8

PEP 8به عنوان محبوب‌ترین راهنمای کدنویسی پایتون، چارچوبی برای نگارش کدهای خوانا و یکنواخت فراهم می‌کند. برخی از مهم‌ترین نکات آن عبارتند از:

نام متغیرها و توابع باید به صورت **snake_case** نوشته شوند.

ثابت‌ها (constants) باید به صورت **snake_case** و با حروف بزرگ نوشته شوند؛ مثلاً :

```
PI = 3.141592
```

انتخاب یک نوع از علامت نقل قول (' یا ") و پایبندی به آن در سراسر پروژه.

تورفتگی (indentation) باید ۴ فاصله باشد؛ توصیه می‌شود از space به جای tab استفاده شود.

خطوط کد نباید از ۷۹ کاراکتر تجاوز کنند.

فاصله قبل از پرانتز بسته و بعد از پرانتز باز الزامی نیست.

عملگرها باید دارای یک فاصله قبل و بعد از خود باشند.

کامنت‌ها باید به صورت جملات کامل نوشته شده و پس از علامت #یک فاصله قرار گیرد.

برای مطالعه کامل استاندارد می‌توانید به PEP 8 مراجعه کنید.

The Zen of Python

یک مجموعه از ۱۹ اصل راهنما برای نوشتن کدهای زیبا و کارآمد در پایتون وجود دارد که در سال ۱۹۹۹ توسط تیم پیترز تدوین شده است. این اصول به عنوان راهنمای کلی برای ایجاد کدهای ساده و خوانا به کار می‌روند:

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.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

برای مشاهده این اصول در محیط پایتون، کافی است دستور زیر را اجرا کنید:

```
import this
```

ساختاردهی کد و ماژولاریتی

یک کد کلین تنها به رعایت قواعد نگارشی محدود نمی‌شود، بلکه سازماندهی صحیح اجزای برنامه نیز نقش مهمی دارد. از اصول مهم در این زمینه می‌توان به موارد زیر اشاره کرد:

تفکیک وظایف (Separation of Concerns):

کدهای مرتبط با یک وظیفه یا منطق خاص در ماژول یا فایل جداگانه نگهداری شوند تا خوانایی و نگهداری آسان‌تر شود.

استفاده از توابع و کلاس‌های کوچک و یکپارچه:

هر تابع یا کلاس باید تنها یک وظیفه مشخص داشته باشد و در صورت نیاز به تغییرات، تنها همان بخش از کد تحت تأثیر قرار گیرد.

ساختار پوشه‌ای منظم:

پروژه‌های پایتون باید دارای ساختاری واضح و منطقی باشند تا درک اجزاء و وابستگی‌های آن برای سایر توسعه‌دهندگان آسان شود.

مدیریت خطا و لاگ‌گذاری

یکی از بخش‌های حیاتی در نوشتن کد کلین، مدیریت مناسب خطاها و ثبت وقایع (لاگ) است:

استفاده از استثناها (Exceptions):

به جای بازگرداندن کدهای خطا، از سازوکار استثنا در پایتون بهره ببرید تا خطاها به صورت کنترل‌شده مدیریت شوند.

لاگ‌گذاری:

استفاده از کتابخانه‌هایی مانند logging برای ثبت وقایع مهم و خطاهای احتمالی می‌تواند در عیب‌یابی و نگهداری کد بسیار مفید باشد.

تست و بازبینی(Refactoring)

کد کلین علاوه بر نگارش اولیه صحیح، نیازمند بهبود و بازبینی مستمر است:

نوشتن تست‌های واحد (Unit Tests):

تست‌های خودکار به اطمینان از عملکرد صحیح کد در مواجهه با تغییرات کمک می‌کنند و از بروز خطاهای ناخواسته جلوگیری می‌کنند.

بازنگری و Refactoring:

کد را به‌طور دوره‌ای مرور کنید و در صورت مشاهده پیچیدگی یا تکرار، آن را ساده‌سازی و بهبود دهید.

کد مرور (Code Review) :

دریافت بازخورد از سایر اعضای تیم می‌تواند به کشف مشکلات و ارائه راهکارهای بهبود کمک شایانی کند.

مستندسازی

نوشتن مستندات مناسب همراه با کد، به درک بهتر عملکرد و کاربرد کد کمک می‌کند:

دسترسی به راهنما و API:

مستندسازی کد با استفاده از ابزارهایی مانند docstring و کتابخانه‌های مستندسازی، به سایر توسعه‌دهندگان کمک می‌کند تا سریع‌تر با کد آشنا شوند.

توضیح چرایی و کاربرد:

مستندات نباید تنها به شرح نحوه استفاده از کد محدود شوند، بلکه باید دلایل طراحی و تصمیمات کلیدی را نیز شرح دهند.

با رعایت اصول کلین کد در پایتون، می‌توان کدهایی نوشت که نه تنها عملکرد مناسبی دارند، بلکه از نظر نگهداری، توسعه و اشتراک‌گذاری نیز بهینه و کارآمد هستند. استفاده از استانداردهای نام‌گذاری، توابع منسجم، کامنت‌گذاری معنادار، مدیریت صحیح خطا و مستندسازی دقیق، پایه‌های ایجاد کدی خوانا و قابل اعتماد را فراهم می‌کنند. این رویکرد در نهایت به بهبود بهره‌وری تیم‌های توسعه و کاهش هزینه‌های نگهداری منجر خواهد شد.