МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

***Циклова комісія програмування***

**РЕФЕРАТ**

з курсу «Основи алгоритмізації та програмування»

на тему:

«Стилі кодування та написання якісного коду»

**Виконали:**

студенти 2 курсу

спеціальності: «ІПЗ»

групи: 2П-20

Ігнатенко Віктор Васильович,

Кучерук Євгеній Романович

**Викладач:**

Марченко С. В

Черкаси-2021

ЗМІСТ:

ВСТУП

1. Стиль Python 4
2. Стандарт оформлення коду 5
3. Чистий код 8
4. Якісний код 10

ВИСНОВОК

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ВСТУП

Норми кодування - це сукупність вказівок, що стосуються певної конкретної мови програмування і встановлюють правила стильового оформлення коду, практики та методи написання програм цією мовою.

Ці норми зазвичай охоплюють організацію файлів, відступи, коментарі, оголошення, інструкції, пропуски, норми найменування, практики та принципи програмування, емпіричні правила програмування, найкращі архітектурні практики та ін. Це поради стосовно структурної якості програмного забезпечення.

Розробникам програмного забезпечення наполегливо рекомендується слідувати цим вказівкам, щоб покращити читабельність їхнього коду та полегшити підтримку програмного забезпечення.

Норми кодування застосовні лише до людей-підтримувачів та рецензентів програмного забезпечення.

Норми можуть бути формалізовані в документованому наборі правил, яких дотримується ціла команда чи компанія, або бути такими ж неформальними, як звичні індивідуальні практики кодування.

Норми кодування не навязуються компіляторами. А отже, не дотримання деяких, чи навіть всіх правил не впливає на ефективність виконання програмного коду.

1.СТИЛЬ PYTHON

Кожна мова програмування має свій стиль і Python не є винятком. Розробники мови є прихильниками певної філософії програмування, яку називають «The Zen of Python» («Дзен Пайтона»). Її текст можна отримати у інтерпретаторі Python за допомогою команди import this.

У більшості мов програмування використовуються спеціальні символи (наприклад, фігурні дужки {}) або ключові слова (наприклад, begin і end) для того, щоб розбити код на частини. У цих мовах хорошим тоном є використання відступів при написанні коду, щоб зробити програму зручною для читання як для себе, так і для інших.

На відміну від багатьох інших мов, Python обов’язково вимагає, щоб блоки коду забезпечувалися відступами.

У Python для побудови структури програми використовуються відступи від лівого краю, які створюються за допомогою пропусків (пробілів).

2.СТАНДАРТ ОФОРМЛЕННЯ КОДУ

Станда́рт оформле́ння ко́ду або станда́рт кодування— набір правил та угод, що використовуються при написанні сирцевого коду на деяких мовах програмування. Стандарт оформлення коду зазвичай приймається та використовується групою розробників програмного забезпечення для єдиного стилю оформлення коду, над яким йде спільна праця. Метою прийняття та використання стандарту є спрощення сприйняття програмного коду людиною, мінімізація навантаження на пам'ять та зір при читанні програми.

Зразком для стандарту кодування може стати набір угод, прийнятих в якій-небудь поширеній печатній праці з мови програмування (наприклад, стандарт кодування мовою С, що отримав коротку назву K&R, виходить з класичного описання С його творцями — Керніганом та Рітчі), поширена бібліотека або API (так, на поширення угорської нотації вплинув той факт, що її використовували в MS-DOS та Windows API, а більшість стандартів кодування для Delphi використовують манеру кодування бібліотеки VCL). Рідше розробник мови програмування випускає детальні рекомендації по кодуванню на ній; випущені, наприклад, стандарти кодування на C# від Microsoft та на Java від Sun. Запропонована розробником та прийнята в загально відомих джерелах манера кодування може бути доповнена та уточнена у корпоративних стандартах.

Стандарт дуже залежить від використовуваної мови програмування. Наприклад, стандарт оформлення коду для С буде сильно відрізнятися від стандарту для мови BASIC. Зазвичай метою стандарту є досягнення такого стану, коли програміст достатньої кваліфікації міг би дати висновки про функцію, яку виконує конкретна ділянка коду, а в ідеалі — також визначити його коректність, вивчивши тільки цю ділянку коду, або мінімально вивчивши інші частини програми. Іншими словами, сенс коду повинен буди зрозумілим з самого коду без необхідності вивчати контекст. Тому стандарти кодування будуються так, щоб шляхом визначеного візуального оформлення елементів програми збільшити інформативність коду для людини.

Зазвичай, стандарт оформлення коду описує:

* засоби вибору значень та використовуваний регістр символів для імен змінних та інших ідентифікаторів.
* запис типу змінної в її ідентифікаторі (угорська нотація).
* регістр символів (нижній, верхній, «верблюжий», «верблюжий» з малої букви), використання знаку підкреслення для розділу слів.
* стиль відступів при оформленні логічних блоків — чи використовуються табуляції, ширина відступу.
* спосіб розстановки дужок, що обмежують логічні блоки.
* використання пробілів при оформленні логічних та арифметичних виразів.
* стиль коментарів та використання документуючих коментарів.

Поза стандартом існують правила про:

* відсутність магічних чисел
* обмеження розміру коду по горизонталі на вертикалі.

Основні принципи поширених стандартів кодування в останній час впливають на синтаксис нових мов програмування. В деяких з них, угоди, що раніше використовувались тільки в стандартах кодування, стають обов'язковими елементами синтаксису. Так, у деяких сучасних мовах (Python, Nemerle) відступи впливають на логіку виконання (тобто блоки коду виділяються не ключовими словами, а розміром відступів). В інших стали частиною мови угоди про регістри літер та префіксів для типів, констант, змінних та полів класів.

Якщо раніше недисциплінований програміст міг ігнорувати стандарти кодування з особистих міркувань, заради зручності та швидкості написання коду, то тепер, при роботі з новими мовами, дотримання стандартів деякою мірою контролюється транслятором.

3.ЧИСТИЙ КОД

Чистий код — це набір правил і принципів, які допомагають полегшити читання, підтримку та розширення нашого коду. Це один з найважливіших аспектів якісного програмного забезпечення. Розробники більше часу читають код, ніж пишуть його, тому важливо, щоб код був якісним.

Код повинен бути простим і вільним від повторюваності. Хоча ми просто надаємо інструкції комп’ютеру, код все одно має бути виразним, тобто легко читатись і чітко повідомляти, для чого він призначений. Написання чистого коду має багато переваг.

Наприклад, такий код:

* легкий для розуміння;
* ефективніший;
* простіший в обслуговуванні, масштабуванні, налагодженні та рефакторингу.

А ще для нього потрібно писати менше документації.

Стандарти написання коду - це збірки правил кодування, настанов та найкращих практик. Кожна мова програмування має власні стандарти, яких слід дотримуватися, щоб писати чистіший код. Зазвичай вони стосуються:

* впорядкування файлів;
* принципів та практик програмування;
* форматування коду (відступів, оголошень, інструкцій);
* конвенцій іменування;
* коментарів.

Щоб сказати, що код чистий і система спроектована грамотно, легкого читання коду недостатньо. Він також повинен мати інші якості:

* Код легко модифікувати. При правильному проектуванні та архітектурі розширення коду обходиться без особливих часових і технічних витрат. Суті коду не мають бути тісно пов'язаними між собою, код має бути частково абстрактним і самодостатнім. Кожна сутність, якою ми оперуємо при розробці, має відповідати тільки за свою частину функціональності.
* Код має бути стабільним, передбачуваним, безпечним і надійним. Яким би простим код не був у читанні, він має бути покритий тестами. Хороший код і тести завжди поруч. Причому важлива не лише кількість тестів, а й їхня якість. Із таким кодом не виникає проблем при запуску та налагодженні, він не викликає змін у навколишньому середовищі.
* Захищений код. При написанні будь-якого коду не можна забувати про загальну безпеку продукту. Рекомендую ознайомитися з базовими принципами безпеки та дотримуватись їх. Якщо йдеться про веб-проекти, рекомендую OWASP.
* Хороший код — код, якого немає. Це не означає, що весь код має бути написаним в один рядок, а ви — обов'язково пишатися тонкими методами. Це означає, що код не варто дублювати, а частина спільних речей має залишитись на рівні абстракцій. Теоретично, спрощення коду має призвести до зменшення кількості дефектів.
* Саме читання коду теж важливе. Кожен розробник має власний стиль написання, а рівень читання залежить від нашого досвіду. Всі ми хочемо писати простий, красивий та лаконічний код.

4.ЯКІСНИЙ КОД

Щоб отримати якісний код, достатньо дотримуватися таких правил:

1.**Дотримуватися стандартів оформлення коду**. У кожної мови програмування є свій стандарт оформлення коду: *які і де робити відступи* (зазвичай 2 або 4 на один рівень вкладення), *де ставити прогалини і дужки, як називати об'єкти, як коментувати код* тощо — див. посилання для C# (1, 2), C++ (1, 2, 3), Java, Javascript (1, 2, 3), Pascal / Delphi, Perl, PHP, Python, Ruby. Деякі організації (наприклад, Google) підлаштовують (змінюють, деталізують) стандарти під свої специфічні потреби. Стандарти можуть містити налаштування редактора коду, які допоможуть дотримуватися відповідного стилю.

2.**Надавати змістовні назви** змінним, функціям і методам. Інакше кажучи, вперше прочитавши код стороння людина має зрозуміти призначення змінної, функції або методу.

*Невдалий щодо назв код:*

const fnm = 'Tom';

const lnm = 'Hanks';

const x = 31;

const l = lstnm.length;

const boo = false;

const curr = true;

const sfn = ‘Remember the name’;

const dbl = [‘1984’, ‘1987’].map((i) => {return i \* 2;});

*Кращий код:*

const firstName = 'Tom';

const lastName = 'Hanks';

const age = 31;

const lastNameLength = lastName.length;

const isComplete = false;

const isCurrentlyActive = true;

const songFileName = ‘Remember the name’;

const yearsDoubled = [‘1984’, ‘1987’].map((year) => {return year \* 2;});

3.**Писати читабельний код**, *який можна однаково легко зрозуміти, незалежно від того, який обсяг цього коду розглядають*.  
  
Наскільки читабельність інтуїтивна? Її можна досягти, якщо дотримуватися деяких правил та угод:

* *cкладні для сприйняття методи мають бути короткими, а прості методи можуть бути довгими;*
* *уникати вкладених циклів і примусового виходу з циклу, де це можливо;*
* *намагатися записувати кожен вираз (вказівку) окремим рядком;*
* *уникати надмірної кількості методів, що послідовно викликають один за одним.*

4.**Будь-яка функція або метод мають виконували лише одну задачу** (принцип поділу обов'язків).

1. function subtract(x, y) {
2. return x - y;
3. }
4. function multiply(x, y) {
5. return x \* y;
6. }
7. function doubleArray(array) {
8. return array.map(number => number \* 2)
9. }

5.**Використовувати коментарі для пояснень** того, що даний метод (процедура) робить, параметрів, значення, що повертають, можливих помилок і виключень. Описати в коментарях роль кожного файлу і класу, вміст кожного поля класу і основні кроки складного коду. Писати коментарі у процесі створення коду, а не після створення. Коментарі мають описувати мету частини коду, а не механізм того, як її досягти. Інакше кажучи, описувати «навіщо», а не «як». При використанні у коментарях назв змінних краще зупинитися і переписати коментар.

*Не правильно:*

countryCode = GetCountryCode(ServerType.BackUp);// Отримати код

if (countryCode == “US”) // Якщо код US

InvokeUSUser(); // Показати користувача

*Правильно:*

// Показати користувачів з US

countryCode = GetCountryCode(ServerType.BackUp);

if (countryCode == “US”)

InvokeUSUser();

6.**Стисло писати код** — див. приклад використання колекції для спрощення коду при створенні списку з одного елемента.

public static List<String> toList(String item) {

List<String> items= new ArrayList<>();

items.add(item);

return items;

}

Компактніше це записують так:

public static List<String> toList(String item) {

return Collections.singletonList(item);

}

7.**Уникати глибоких вкладень**, які ускладнюють сприйняття коду і виявлення помилок.

*Не правильно:*

public int DoSmth(){

if (IsWritable(folder)) {

if (fp == FileOpen(filePath,”w”)) {

if (stuff = getSomeStuff()) {

if (FileWrite(filePath,stuff)){ // ..

} else {

return false;

}

} else {

return false;

}

} else {

return false;

}

} else {

return false;

}

}

*Правильно:*

public int DoSmth(){

if (!IsWritable(folder))

{

return false;

}

if (fp != FileOpen(filePath, “w”))

{

return false;

}

if (stuff != getSomeStuff())

{

return false;

}

if (!FileWrite(filePath, stuff))

{ //..

}

else return false;

}

8.**Розділяти код на короткі частини.** Код кожного методу, функції чи блоку має не виходити за межі вікна (25−50 рядків). Інакше його потрібно поділити на коротші частини. Хоча б порожніми рядками. Призначення кожного блоку бажано описати у коментарі на початку кожного блоку. Якщо частини коду виконують різнорідні завдання, то його потрібно розділити відповідним чином.

9.**Використовувати тестування частинами** (Unit-тести). Складність сучасного програмного забезпечення робить його створення дорожче, а тестування важче. Продуктивним підходом буде супровід кожної частини коду тестами, які перевіряють правильність саме його роботи. Цей підхід спрощує налагодження, бо дозволяє виявити помилки раніше. Unit-тестування особливо необхідно, коли програмують інтерпретованою мовою з динамічною типізацією (наприклад, JavaScript, Python чи Ruby), бо у цьому випадку можна виявити будь-які помилки лише на етапі виконання. Для мови зі статичною типізацією (наприклад C#, C++ чи Java) частину помилок можна виявити під час компіляції.

ВИСНОВОК

Писати чистий код складно. Немає єдиного плану, щоб все було добре: для майстерності потрібен час і досвід. Ми розглянули деякі стандарти та загальні вказівки, які можуть допомогти написати кращий код. Потрібно бути послідовним і намагатися писати простий код, який легко тестувати. Якщо код важко тестувати, його, ймовірно, важко використовувати.

Якісне програмне забезпечення не обходиться без тестів. Тестування програмного забезпечення дозволяє нам виявляти вади та помилки до розгортання ПЗ. Тести такі ж важливі, як і робочий код, і потрібно витратити достатньо часу на роботу з ними.

Помилки в коді чимось схожі на вуглецевий слід. Цілком уникнути його неможливо, а зайвий вихлоп сам собою не вб'є ані людства, ані природи. Проте зниження негативного ефекту від свого перебування на планеті сьогодні є природною потребою. Приблизно так само написання чистого коду є відповідальністю кожного розробника. Незалежно від того, який саме шлях ви виберете, необхідно прагнути писати працюючий і зрозумілий код.

Добре, якщо вийде не перетворювати чистоту на фетиш, враховуючи термін життя нашого коду та оцінюючи доцільність подальших поліпшень. Головне пам'ятати про людей: користувачів, яких може підвести раптова відмова навіть невеликої частини розробленої нами системи, та інженерів, які мають цю систему підтримувати.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стиль Python. URL: https://pythonguide.rozh2sch.org.ua/ (Дата звернення: 10.01.2022)
2. Стиль коду. URL: https://uk.xn----7sbiewaowdbfdjyt.pp.ua/ (Дата звернення: 10.01.2022)
3. Стандарт оформлення коду. URL: https://uk.wikipedia.org/ (Дата звернення: 10.01.2022)
4. Стиль коду. URL: https://amp.uk (Дата звернення: 10.01.2022)
5. Чистий код. URL: https://www.dataart.com.ua/ (Дата звернення: 10.01.2022)
6. Якісний код. URL: http://www.kievoit.ippo.kubg.edu.ua/ (Дата звернення: 10.01.2022)
7. Ідеальний код. URL: https://dou.ua/ (Дата звернення: 10.01.2022)
8. Чистий Python-код. URL: https://codeguida.com/ (Дата звернення: 10.01.2022)
9. Правила написання якісного коду. URL: https://studopedia.com.ua/ (Дата звернення: 10.01.2022)