Міністерство освіти і науки, молоді та спорту України

Львівський національний університет імені Івана Франка

Факультет електроніки

Кафедра радіофізики

Курсова робота

з курсу “ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ”

на тему:

« ДОСЛІДЖЕННЯ ОБ’ЄКТНО ОРІЄНТОВАНИХ МОЖЛИВОСТЕЙ МОВИ PYTHON »

Виконала студентка

групи ФЕІ-22

\_\_\_\_\_\_Литвин Віра

Науковий керівник:

\_\_\_доц. Хвищун І. О.

Львів 2011

**ЗМІСТ**

Анотація……………………………………………………………………..

ВСТУП………………………………………………………………………

1. ООП………………………………..…

1.1.Передумови виникнення…………………………………………….…

1.2. Опис парадигми ………………………………….….

2. Мова PYTHON ……………...

2.1. Загальний огляд мови …………………

2.2. Оптимізація PYTHON-коду.………………………..

2.3. Порівняння з іншим мовами

2.3.1. Вплив інших мов

2.3.2. Недоліки

2.3.2.1.Низька швидкодія

2.3.2.2. Відсутність статичної типізації

2.3.2.3. Неможливість модифікації вбудованих класів

2.3.2.4. Глобальне блокування інтерпретатора

2.4. Особливості використання

2.4.1. Графічні бібліотеки

2.4.2. Стандартна бібліотека

3. Програмування на мові PYTHON …..……………..

3.1. Базові поняття …………………………………….…

3.1.1. Оператори

3.1.2.Вирази

3.1.3. Імена

3.1.4. Рядки документації

3.1.5. Директиви

3.1.6. Інтерактивний режим

3.1.7.Керування контекстом виконання

3.1.8. Обробка винятків

3.2. Функціональне програмування

3.3. Об’єктно-орієнтоване програмування

3.3.1. Можливості та особливості

3.3.2. Модулі та пакети

3.3.3. Інтроспекція

3.3.4. Ітератори

3.3.5. Генератори

3.3.6. Декоратори

3.3.7. Інші можливості

3.2. Відмінності від інших мов …………………………………………….

3.3. Особливості реалізації ……………………………………………….....

4. Google Docs ……………………………………………………………

4.1. Google Docs …………………………………………………………….

4.1.1. Writely…………………………………………………………..

4.1.2. Google Spread Sheets…………………………………………….

4.1.3. Google Presentations ……………………………………………

4.1.4. Історія ………………………………………………………….

4. Практична частина……………………………………………………

5.1. Інтерпретація результатів ……………………………………………..

5.2. Розробка програми …………………………………………………….

5.2.1. Проектування інтерфейсу ……………………………………

5.2.2. Компілювання програми ……………………………………..

ВИСНОВОК……………………………………………………………..

Використана література … …………………………………………

ДОДАТОК………………………………………………………….

Додаток 1. Код програми ………………………………………………

Додаток 2. Код інтерфейсу …………………………………………….

**Анотація**

У роботі проведено дослідження об’єктно-орієнтованого підходу у програмуванні на основі вивчених мов, а також нової для нас мови програмування Python. Тестову програму на цій мові, що демонструє деякі принципи ООП реалізовано в середовищі Python IDLE з використанням графічної оболонки Gui а також за допомогою бібліотек PyQt4

Зроблено висновки про ефективність виконаної програми, переваги вивченої мови та особливості електронних інтернет – таблиць Google Spread Sheets.

**Вступ**

Останнім часом ідея об’єктно-орієнтованого програмування (ООП) – кардинально нова ідеологія написання програм, все більше займає розуми програмістів.

Об’єктно-орієнтовані програми більш прості й мобільні, їх легше модифікувати та супроводжувати, ніж їх “традиційних” побратимів. Крім того, схоже, сама ідея об’єктної орієнтованості при грамотному її використанні дозволяє програмі бути навіть більш захищеною від різного роду помилок, ніж це передбачав програміст в момент роботи над нею.

**Актуальність** такого дослідження зумовлена дедалі більшою зацікавленістю програмістів мовою PYTHON, завдяки легкості її освоєння, багатофункціональності, а, особливо, через кросплатформенність.

**Метою** роботи є дослідити об’єктно-орієнтовані можливості мови PYTHON.

Відповідно до мети дослідження, ми ставимо перед собою такі **завдання**:

* ознайомитись з синтаксисом та філософією PYTHON;
* дослідити об’єктно-орієнтовані можливості мови;
* порівняти їх з вивченими досі особливостями ООП в Object Pascal, Delphi, C++ та C#;
* написати просту програму для демонстрації цих можливостей.

**Матеріалом** роботи стали Python2.7. , PyQt4 та Google Docs.

**Практична цінність** роботи полягає в тому, що її матеріали дослідження можуть бути використані для подальшого, більш глибокого вивчення даної теми, результати практичної роботи можуть застосовуватись багатьма користувачами для задоволення щоденних потреб.

**1. ООП**

**1.1. Передумови виникнення**

Розвиток у галузі програмування безпосередньо залежить від задач, які пропонуються для вирішення з його допомогою. Складність задач у програмуванні безперервно зростає.

Загальноприйняті у певний період часу підходи та стилі програмування утворюють так звані парадигми програмування. Можна виділити наступну послідовність парадигм програмування:

Процедурне – Структурне – Модульне – Об’єктно-орієнтоване.

Поява нової парадигми сприяє розробці нових мов програмування та інструментальних засобів.

З розвитком програмування виникла ідея поєднати в межах однієї сутності дані і код, що безпосередньо опрацьовує ці дані. Така сутність отримала назву об’єкт, а відповідний підхід до створення програм називають об’єктно-орієнтованим програмуванням.

З погляду сприйняття людиною об’єктом може бути:

- відчутний і (або) видимий предмет;

- щось, що сприймається мисленням;

- щось, на що спрямовано думку або дію.

Кожен об’єкт має суттєві характеристики, які відрізняють його від усіх інших об’єктів. Сукупність таких характеристик називається абстракцією. Розрізняють абстракції стану та поведінки об’єкта.

Стан(дані об’єкта) характеризується переліком та значеннями певних ознак.

Поведінка (код об’єкта) визначається набором операцій, які виконуються об’єктом, або над об’єктом.

Кожен об’єкт є екземпляром (представником) певного класу. Відповідно, клас – це певна множина об’єктів, що мають спільні характеристики стану та поведінки.

Відповідно до парадигми об'єктно-орієнтованого програмування, кожний об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

Незважаючи на те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до1990-тих.

**1.2. Опис парадигми**

Об’єктно-орієнтоване програмування засноване на таких принципах:

- моделювання предметів та явищ реального світу;

- наявність типів даних, що створюються користувачем (класи);

- приховування деталей реалізації (інкапсуляція);

- можливість повторного використання коду (наслідування);

- інтерпретація викликів процедур та функцій на етапі виконання(поліморфізм).

**Інкапсуляція**(англ. encapsulation) – спосіб поєднати дані і код, який опрацьовує ці дані. При цьому частина даних або коду може бути захищена від зовнішнього впливу (поза об’єктом) або помилкового використання.

**Наслідування**(англ. inheritance)**.** На основі одного класу можна створити інший, який буде використовувати (успадковувати) стан та поведінку першого. У цьому випадку перший клас називається батьківським (базовим, надкласомабо предком), а другий – дочірнім (похідним, підкласом або нащадком). Окрім використання полів та методів батьківського класу, дочірній клас може їх змінювати або доповнювати, а також мати власні.

**Поліморфізм** (англ. polymorphism) – можливість позначати одним ім’ям об’єкти різних класів, які мають спільного предка. Тоді коженоб’єкт, що позначається таким поліморфним ім’ям, може по-своєму реагувати на якийсь спільний набір операцій.

На думку [Алана Кея](http://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B0%D0%BD_%D0%9A%D0%B5%D0%B9), розробника мови [Smalltalk](http://uk.wikipedia.org/wiki/Smalltalk), якого вважають одним з «батьків-засновників» ООП, об'єктно-орієнтований підхід полягає в наступному наборі основних принципів:

* Все є об'єктами.
* Всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, при якій один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи і отримуючи повідомлення. Повідомлення — це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися при виконанні дії.
* Кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів.
* Кожен об'єкт є представником (екземпляром, примірником) класу, який виражає загальні властивості об'єктів.
* У класі задається поведінка (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії.
* Класи організовані у єдину деревовидну структуру з загальним корінням, яка називається ієрархією [успадкування](http://uk.wikipedia.org/wiki/%D0%A3%D1%81%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

Стратегію ООП найкраще описати як зміщення пріоритетів у процесі програмування від функціональності програми до структур даних. Це дозволяє програмісту моделювати в створюваних додатках реальні об’єкти і ситуації. Технологія ООП володіє трьома головними перевагами:

* вона проста для розуміння: ООП дозволяє мислити категоріями повсякденних об’єктів
* забезпечує підвищену надійність і проста для супроводу – правильне проектування забезпечує простоту розширення і модифікації об’єктно-орієнтованих програм. Модульна структура дозволяє вносити незалежні зміни в різні частини програми, зводячи до мінімуму ризик помилок програмування
* прискорює цикл розробки – модульність і тут грає важливу роль, оскільки різні компоненти об’єктно-орієнтованих програм можна легко використовувати в інших програмах, що зменшує надмірність коду і знижує ризик занесення помилок при копіюванні

Специфіка ООП помітно підвищує ефективність праці програмістів і дозволяє їм створювати більш потужні, масштабовані та ефективні програми.

**2. Мова PYTHON**

**2.1. Загальний огляд мови**

**Python** (рекомендоване прочитання — « **Па́йтон** », запозичено назвуз британського шоу Монті Пайтон) — інтерпретована об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою.

Розроблена в 1990 році Гвідо ван Россумом.

Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонент.

Пайтон підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду.

Інтерпретатор Пайтон та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на всіх основних платформах.

В мові програмування Пайтон підтримується декілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Серед основних переваг Python можна виділити такі:

* чистий синтаксис (для виділення блоків слід використовувати пробіли);
* портабельність (що властиве більшості інтерпретованих мов);
* стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
* можливість використання Пайтона в діалоговому режимі (дуже корисне для експериментування та вирішення простеньких задач);
* стандартний дистрибутив має простеньке, але разом досить потужне середовище розробки, яке зветься IDLE і яке написане мовою Пайтон;
* зручний для вирішення математичних проблем (розуміє комплексні числа, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор).

Його легко використовувати, у той же час це справжня мова програмування, що має значно багатшу структуру та підтримку для написання великих програм, ніж традиційна оболонка.

З іншого боку, Пайтон пропонує набагато кращу перевірку помилок ніж С, і, будучи мовою *дуже високого рівня*, він має вмонтовані типи даних високого рівня, зокрема гнучкі масиви та словники, для ефективного створення яких на мові С потрібно кілька днів.

Завдяки загальнішім типам даних, Пайтон може бути застосований для вирішення набагато ширшого кола проблем, ніж така мова як AWK, чи навіть Perl; водночас багато речей можуть бути створені на Пайтоні так само просто, як і на цих мовах.

Пайтон дозволяє розбити вашу програму на модулі, які можуть згодом використовуватися в інших програмах, написаних на Пайтоні. Пайтон має велику кількість стандартних модулів, які ви можете покласти в основу своїх програм або на яких можете почати вчитися програмувати. Існують також вбудовані модулі для таких речей як файловий ввід-вивід, системні виклики, сокети, і навіть графічні інтерфейси.

Пайтон - це інтерпретована мова, що може зберегти вам чимало часу при розробці програм, тому що компіляція та прив'язування (linking) непотрібні. Інтерпретатор може також використовуватися у діалоговому режимі, що спрощує експериментування з різними рисами мови, написання одноразових програм чи тестування шляхом зворотньої розробки. Це також зручний настільний калькулятор.

Пайтон дозволяє створювати дуже компактні та читабельні програми. Типова програма, написана на Пайтоні - набагато коротша ніж еквівалентна програма на C чи C++, що відбувається з таких причин:

* структури даних високого рівня дозволяють виразити складні операції за допомогою окремих тверджень;
* групування тверджень робиться за допомогою виділення пробілами замість фігурних дужок;
* декларація змінних чи аргументів непотрібна;

Пайтон *можна розширити*: якщо ви вмієте програмувати на C, то вам буде легко додати нову вбудовану функцію до інтерпретатора, що надасть можливість виконувати критичні операції з максимальною швидкістю, або прив'язати Пайтон до вже скомпільованих бібліотек (напр., спеціальні графічні бібліотеки). Також ви можете прив'язяти інтерпретатор Пайтона до програми, написаної на C і використовувати його як розширення чи командну мову для вашої програми.

На основі Python було створено кілька спеціалізованих підмножин мови, в основному призначених для статичної компіляції в машинний код. Деякі з них:

* RPython — створена в рамках проекту PyPy значно обмежена реалізація Python без динамізму часу виконання та деяких інших можливостей. RPython код можна компілювати в безліч інших мов/платформ — C, JavaScript, Lisp, .NET, LLVM. На RPython написаний інтерпретатор PyPy.
* Pyrex — обмежена реалізація Python, але трохи менше, ніж RPython. PyReX розширено можливостями статичної типізації типами з мови С і він дозволяє вільно змішувати типізований та не типізований код. Призначений для написання модулів розширень, компілюється в код на мові С.
* Cython — розширена версія Pyrex.
* pyastra — компілятор Python коду в асемблер для PIC архітектури.
* shed-skin — призначений для компіляції неявно статично типізованого Python коду в оптимізований код на мові С++, проект далекий від завершення.

**2.2. Оптимізація Python - коду**

У порівнянні з Ruby та деякими іншими мовами, в Python відсутня можливість модифікувати вбудовані класи, такі, як int, str, float, list та інші, що, однак, дозволяє Python споживати менше оперативної пам'яті і швидше працювати. Ще однією причиною введення такого обмеження є необхідність узгодження з модулями розширення. Багато модулів (з метою оптимізації швидкодії) перетворять Python-об'єкти елементарних типів до відповідних Сі-типів замість маніпуляцій з ними за допомогою Сі-API.

У стандартній бібліотеці Python є профайлер (модуль **profile**), який можна використовувати для збору статистики про час роботи окремих функцій. Для вирішення питання про те, який варіант коду працює швидше, можна використовувати модуль timeit.

Як і в будь-якій мові програмування, у Python є свої прийоми оптимізації коду. Оптимізувати код можна, виходячи з різних (часто конкуруючих між собою) критеріїв (збільшення швидкодії, зменшення обсягу необхідної оперативної пам'яті, компактність вихідного коду і т. д.). Найчастіше програми оптимізують за часом виконання.

Тут є кілька очевидних правил:

* Не потрібно оптимізувати програму, якщо швидкість її виконання достатня.
* Використовуваний алгоритм має певну тимчасову складність, тому перед оптимізацією коду програми варто спочатку переглянути алгоритм.
* Варто використовувати готові і налагоджені функції та модулі, навіть якщо для цього потрібно небагато обробити дані. Наприклад, у Python є вбудована функція **sort()**.
* Профілювання допоможе з'ясувати вузькі місця. Оптимізацію потрібно починати з них.

Python має такі особливості і пов'язані з ними правила оптимізації:

* Виклик функцій є досить дорогою операцією, тому всередині вкладених циклів потрібно прагнути уникати виконання функцій або, наприклад, переносити цикл у функції. Функція, обробна послідовність, ефективніша, ніж обробка тієї ж послідовності у циклі з викликом функції.
* Намагайтеся винести з глибоко вкладеного циклу все, що можна обчислити в зовнішніх циклах. Доступ до локальних змінних більш швидкий, ніж до глобальних, або ніж доступ до полів.
* Оптимізатор **psyco** може допомогти прискорити роботу модулю програми за умови, що модуль не використовує динамічних властивостей мови Python.
* У випадку, якщо модуль проводить масивну обробку даних і оптимізація алгоритму та коду не допомагає, можна переписати критичні ділянки, скажімо, на мові Сі або Cython.

Інструмент під назвою Pychecker допоможе проаналізувати вихідний код на Python та видати рекомендації згідно зі знайденими проблемами (наприклад, невикористані імена, зміна сигнатури методу при його перевантаженні тощо). В ході такого статичного аналізу вихідного коду можуть бути виявлені і помилки.

Pylintпокликаний вирішувати близькі завдання але більш схиляється до перевірки стилю коду.

**2.3. Порівняння з іншими мовами**

Найчастіше Python порівнюють з Perl та Ruby. Ці мови також є інтерпретованими та мають приблизно однакову швидкість виконання програм. Як і Perl, Python може успішно застосовуватися для написання скриптів (сценаріїв).

Як і Ruby, Python є добре продуманою системою для ООП.

Засоби функціонального програмування частково запозичені з Scheme та Icon.

У середовищі комерційних застосунків швидкість виконання програм на Python можуть порівнюювати з Java-застосунками.

Незважаючи на те, що Python має досить самобутній синтаксис, одним із принципів дизайну цієї мови є принцип найменшого подиву.

**2.3.1.Вплив інших мов**

З'явившись порівняно пізно, Python створювався під впливом багатьох мов програмування:

* [ABC](http://uk.wikipedia.org/w/index.php?title=ABC_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)&action=edit&redlink=1) — поле для угруповання операторів, високорівневі структури даних (map) (фактично, Python створювався як спроба виправити помилки, допущені при проектуванні ABC);
* [Modula-3](http://uk.wikipedia.org/w/index.php?title=Modula-3&action=edit&redlink=1) — пакети, модулі, використання else спільно з try та except, іменовані аргументи функцій (на це також вплинув [Common Lisp](http://uk.wikipedia.org/wiki/Common_Lisp));
* [Сі](http://uk.wikipedia.org/wiki/%D0%A1%D1%96_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [C++](http://uk.wikipedia.org/wiki/C%2B%2B)— деякі синтаксичні конструкції (як пише сам [Гвідо ван Россум](http://uk.wikipedia.org/wiki/%D0%93%D0%B2%D1%96%D0%B4%D0%BE_%D0%B2%D0%B0%D0%BD_%D0%A0%D0%BE%D1%81%D1%81%D1%83%D0%BC) — він використовував найбільш несуперечливі конструкції з С, щоб не викликати неприязнь у Сі-програмістів до Python);
* [Smalltalk](http://uk.wikipedia.org/wiki/Smalltalk) — об'єктно-орієнтоване програмування;
* [Lisp](http://uk.wikipedia.org/wiki/Lisp) — окремі риси [функціонального програмування](http://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) (lambda, map, reduce, filter та інші);
* [Fortran](http://uk.wikipedia.org/wiki/Fortran) — зрізи масивів, комплексна арифметика;
* [Miranda](http://uk.wikipedia.org/w/index.php?title=%D0%9C%D1%96%D1%80%D0%B0%D0%BD%D0%B4%D0%B0_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)&action=edit&redlink=1) — спискові вирази;
* [Java](http://uk.wikipedia.org/wiki/Java) — модулі logging, unittest, threading (частина можливостей оригінального модуля не реалізована), xml.sax стандартної бібліотеки, спільне використання finally та except при обробці виключень, використання @ для декораторів;
* [Icon](http://uk.wikipedia.org/w/index.php?title=Icon_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)&action=edit&redlink=1) — [генератори](http://uk.wikipedia.org/wiki/Python#.D0.93.D0.B5.D0.BD.D0.B5.D1.80.D0.B0.D1.82.D0.BE.D1.80.D0.B8).

Більша частина інших можливостей Python (наприклад, байт-компіляція вихідного коду) також була реалізована раніше в інших мовах.

**2.3.2.Недоліки**

#### 2.3.2.1.Низька швидкодія

Пайтон, як і багато інших інтерпретованих мов, не застосовують, наприклад, JIT-компілятори, мають загальний недолік — порівняно невисоку швидкість виконання програм.Однак, у випадку з Python цей недолік компенсується зменшенням часу розробки програми. У середньому, програма, написана на Python, в 2-4 рази компактніше, ніж її аналог на C++ або Java. Збереження байт-коду (файли \*.pyc та \*.pyo) дозволяє інтерпретатору не витрачати зайвий час на перекомпіляцій коду модулів при кожному запуску, на відміну, наприклад, від мови Perl. Крім того, існує спеціальна JIT-бібліотека psyco, що дозволяє прискорити виконання програм (проте призводить до збільшення споживання оперативної пам'яті). Ефективність psyco сильно залежить від архітектури програми.

Існують проекти реалізацій мови Python, що вводять високопродуктивні віртуальні машини (ВМ) як компілятори заднього плану. Прикладами таких реалізацій може служити PyPy, що базується на LLVM; більш ранньої ініціативою є проект Parrot. Очікується, що використання ВМ типу LLVM призведе до тих самих результатів, що й використання аналогічних підходів для реалізацій мови Java, де низька обчислювальна продуктивність в основному подолана.

Безліч програм/бібліотек для інтеграції з іншими мовами надають можливість використовувати іншу мову для написання критичних ділянок.

У найпопулярнішій реалізації мови Python інтерпретатор досить великий і більш вимогливий до ресурсів, ніж в аналогічних популярних реалізаціях Tcl, Forth, LISP або Lua, що обмежує його застосування у вбудованих системах. Тим не менше, Python знайшов застосування в КПК і деяких моделях мобільних телефонів.

#### 2.3.2.2.Відсутність статичної типізації

Відсутність статичної типізації є не стільки вадою інтерпретатора, скільки вибором розроблювача мови. Справа в тому, що в Python прийнята так звана «Качина типізація». У силу цього типи переданих значень недоступні на етапі компіляції, та помилки на зразок AttributeError можуть виникати під час виконання. Відсутність статичної типізації також є однією з основних причин низької швидкодії.

Існують модулі, які дозволяють контролювати типи параметрів функцій на етапі виконання, наприклад *typecheck* або *method signature checking decorators*. При цьому, однак, безпосередньо інтерпретатор не буде перевіряти типи, а тільки додавати відповідну інформацію до метаданих функції для її (інформації) подальшого використання модулями розширень.

Відсутність статичної типізації і деякі інші причини не дозволяють реалізувати в Python механізм перевантаження функцій на етапі компіляції.

Можливості Python дозволяють реалізувати динамічне перевантаження на етапі виконання, що, звичайно, уповільнює виклик, оскільки з’ясування питання, яку саме функцію викликати проводиться при кожному зверненні і є, в загальному випадку, досить складною процедурою. Відсутність перевантаження в Python намагаються компенсувати використанням віртуальних функцій.

len = **lambda** x : x.\_\_len\_\_() *# це лише приклад*

Перевантаження функцій реалізовано за допомогою різних сторонніх бібліотек, в тому числі PEAK має надзвичайно багато можливостей для реалізації механізму перевантаження функцій з використанням довільних правил.

#### 2.3.2.3.Неможливість модифікації вбудованих класів

У порівнянні з Ruby та деякими іншими мовами, в Python відсутня можливість модифікувати вбудовані класи, такі, як int, str, float, list та інші, що, однак, дозволяє Python споживати менше оперативної пам'яті і швидше працювати. Ще однією причиною введення такого обмеження є необхідність узгодження з модулями розширення. Багато модулів (з метою оптимізації швидкодії) перетворюють Python-об'єкти елементарних типів до відповідних Сі-типів замість маніпуляцій з ними за допомогою Сі-API.

#### 2.3.2.4.Глобальне блокування інтерпретатора (GIL)

GIL (Global Interpreter Lock) — проблема, притаманна CPython, Stackless та PyPy, але відсутня в Jython та IronPython.

При своїй роботі основний інтерпретатор Python постійно використовує велику кількість нитко-небезпечних даних. В основному це словники, в яких зберігаються атрибути об'єктів. Для уникнення руйнування цих даних при спільній модифікації з різних ниток перед початком виконання декількох інструкцій (за замовчуванням 100) нитка інтерпретатора захоплює GIL, а після закінчення звільняє.

Внаслідок цієї особливості в кожен момент часу може виконуватися тільки одна нитка Python текстів, навіть якщо на комп'ютері є кілька процесорів або процесорних ядер (GIL також звільняється на час виконання блокуючих операцій, таких як введення-виведення, зміни/перевірка стану синхронізуючих примітивів та інших — таким чином, якщо одна нитка блокується, інші можуть виконуватися).

Була зроблена спроба переходу до більш гранульованої синхронізації, проте через часті захоплення/звільнення блокувань ця реалізація виявилася занадто повільною.

У найближчому майбутньому перехід від GIL до інших технік не передбачається, однак є python-safethread — CPython без GIL і з деякими іншими змінами (за твердженнями його авторів, на однониткових застосунках швидкість відповідає 60-65% від швидкості на оригінальному CPython).

Ця проблема має два основних варіанти вирішення.

Перший — відмова від спільного використання змінюваних даних. При цьому дані дублюються в нитках і необхідність забезпечення їхньої синхронізації (якщо така потрібна) лягає на програміста. Цей підхід веде до збільшення споживання оперативної пам'яті (однак не настільки сильно, як при використанні процесів).

Другий підхід — забезпечення більш гранульованої синхронізації — для окремих структур даних. У цьому випадку падає продуктивність внаслідок збільшення числа звільнень/захоплень блокувань.

Якщо необхідно паралельне виконання декількох ниток Python-коду, то можна скористатися процесами, наприклад, модулем processing, який імітує семантику стандартного модуля threading, але використовує процеси замість ниток. Є безліч модулів, що спрощують написання паралельних та/або розподілених за стосунків на Python, таких як parallelpython, Pypar, pympi та інші. GIL звільняється при виконанні коду більшості розширень, наприклад, NumPy/SciPy, дозволяючи на час розрахунків виконуватися іншій Python-нитці.

Іншим рішенням може бути використання IronPython або Jython, позбавлених даного недоліку.

**2.4.Особливості використання**

Пайтон портований на всі відомі платформи — від КПК до мейнфреймів. Існують порти під [Windows](http://uk.wikipedia.org/wiki/Windows), всі варіанти [UNIX](http://uk.wikipedia.org/wiki/UNIX) (включно з [Linux](http://uk.wikipedia.org/wiki/Linux)), [Plan 9](http://uk.wikipedia.org/wiki/Plan_9), [Mac OS](http://uk.wikipedia.org/wiki/Mac_OS) і [Mac OS X](http://uk.wikipedia.org/wiki/Mac_OS_X), [Palm OS](http://uk.wikipedia.org/wiki/Palm_OS),[OS/2](http://uk.wikipedia.org/wiki/OS/2), Amiga, AS/400 і навіть OS/390 і [Symbian](http://uk.wikipedia.org/wiki/Symbian).

При цьому, на відміну від багатьох портованих систем, на кожній платформі Пайтон підтримує характерні для даної платформи технології (наприклад, [Microsoft](http://uk.wikipedia.org/wiki/Microsoft) [COM](http://uk.wikipedia.org/wiki/COM)). Крім того, існує спеціальна версія Пайтона для віртуальної машини [Java](http://uk.wikipedia.org/wiki/Java) — [Jython](http://uk.wikipedia.org/wiki/Jython), що дозволяє інтерпретатору виконуватися на будь-якій системі, що підтримує Java, класи Java можуть безпосередньо використовуватися з Пайтона і навіть бути написаними на Пайтоні. Нещодавно почалася розробка системи, спрямованої на більш повну інтеграцію з платформою [.NET](http://uk.wikipedia.org/wiki/.NET) — [Iron Python](http://uk.wikipedia.org/w/index.php?title=IronPython&action=edit&redlink=1).

### 2.4.1 Графічні бібліотеки

З Python поставляється бібліотека tkinter на основі Tcl/Tk для створення крос-платформних програм з графічним інтерфейсом.

Для науково-технічної мети найбільшого поширення набуло використання matplotlib — бібліотеки з інтерфейсом, аналогічним MATLAB Plot Tool.

Існують розширення, що дозволяють використовувати всі основні GUI бібліотеки — wxPython, засноване на бібліотеці wxWidgets, PyGTK для GTK+, PyQt та PySide для Qt та інші. Деякі з них також надають широкі можливості для роботи з базами даних, графікою та мережами, використовуючи всі можливості бібліотеки, на якій базуються.

Для створення ігор та програм, що вимагають нестандартного інтерфейсу, можна використовувати бібліотеку Pygame. Вона також надає великі засоби роботи з мультимедіа: з її допомогою можна керувати звуком і зображеннями, відтворювати відео. Надаване pygame апаратне прискорення графіки OpenGL має більш високорівневий інтерфейс в порівнянні з PyOpenGL, що копіює семантику С-бібліотеки для OpenGL. Є також PyOgr, що забезпечує прив'язку до Ogre — високорівневої об'єктно-орієнтованої бібліотеки 3D-графіки. Крім того, існує бібліотека pythonOCC, що забезпечує прив'язку до середовища 3D-моделювання та симуляції OpenCascade.

Для роботи з растровою графікою використовується бібліотека Python Imaging Library.

### 2.4.2. Стандартна бібліотека

Python поставляється «з батарейками в комплекті».

Багата стандартна бібліотека є однією з привабливих сторін Python. Тут є засоби для роботи з багатьма мережевими протоколами та форматами Інтернету, наприклад, модулі для написання HTTP-серверів та клієнтів, для розбору та створення поштових повідомлень, для роботи з XML тощо Набір модулів для роботи з операційної системою дозволяє писати крос-платформні застосунки. Існують модулі для роботи з

* + регулярними виразами,
  + текстовими кодуваннями,
  + мультимедійними форматами,
  + криптографічними протоколами,
  + архівами,
  + серіалізацією даних,
  + підтримки юніт-тестування та ін.

**3. Програмуваня на мові PYTHON**

**3.1.Базові поняття**

Пайтон підтримує *динамічну типізацію*, тобто, тип змінної визначається лише під час виконання. З базових типів слід зазначити підтримку *цілих чисел довільної довжини* і *комплексних чисел*.

Пайтон має багату бібліотеку для роботи з *рядками*, зокрема, кодованими в юнікоді.

З колекцій Пайтон підтримує кортежі (*tuples*), списки (масиви), словники (асоціативні масиви) і від версії 2.4, множини.

Система класів підтримує множинне успадкування і метапрограмування. Будь-який тип, включаючи базові, входить до системи класів, й за необхідності можливе успадкування навіть від базових типів.

Програми, написані на Пайтоні, легко читаються. Мова має чіткий і послідовний синтаксис, продуману модульність і маштабованість.

Пайтон пропонує зручну систему автоматичного документування коду — кожен [модуль](http://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C), [клас](http://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), або [функція](http://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F) може мати [рядок документації](http://uk.wikipedia.org/w/index.php?title=%D0%A0%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D1%96%D1%97&action=edit&redlink=1) (*docstring*). При інтерактивній роботі рядки документації доступні із системи допомоги.

**3.1.1.Оператори**

Набір операторів досить традиційний. Ось деякі з них:

* Умовний оператор if (якщо). Альтернативний блок після else (інакше). Якщо умов і альтернатив кілька, можна використовувати elif (скор. Від else if).
* Оператори циклу while (поки що) та for (для). Всередині циклу можливе застосування break та continue для переривання циклу, і переходу відразу до наступної ітерації відповідно.
* Оператор визначення класу class.
* Оператор визначення функції, методу або генератора def. Всередині можливе застосування return (повернення) для повернення з функції або методу, а у разі генератора — yield (давати).
* Оператор обробки винятків try-except-else або try-finally (починаючи з версії 2.5, можна використовувати finally, except та else в одному блоці).
* Оператор pass нічого не робить. Використовується для порожніх блоків коду.

Однією з цікавих синтаксичних особливостей мови є виділення блоків програми з допомогою відступів ([пробілів](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B1%D1%96%D0%BB) чи [табуляцій](http://uk.wikipedia.org/wiki/%D0%A2%D0%B0%D0%B1%D1%83%D0%BB%D1%8F%D1%86%D1%96%D1%8F)), у Пайтоні відсутні операторні дужки *begin/end* як у мові [Паскаль](http://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_Pascal) чи фігурні дужки, як у [C](http://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)).

Позаяк у більшості стилів форматування операторні дужки займають цілий рядок, такий «трюк» дозволяє скоротити кількість рядків і символів в програмі, та навчає «доброго» стилю програмування. Нище наведено приклад для порівняння.

|  |  |
| --- | --- |
| **Програма на Сі** | **Еквівалентна програма на Python** |
| unsigned factorial (unsigned x)  {  if (x == 0)  {  return 1;  }  else  {  return x \* factorial(x-1);  }  } | **def** factorial(x):  **if** x == 0:  **return** 1  **else**:  **return** x \* factorial(x-1) |

Отже, поведінка і навіть коректність програми може залежати від початкових відступів у тексті. Деякі критики мови вважають таку поведінку протиінтуїтивною.

**3.1.2. Вирази**

Вираз є повноправним оператором в Python. Склад, синтаксис, асоціативність і пріоритет операцій досить звичні для мов програмування та покликані мінімізувати вживання дужок.

Окремо варто згадати *операцію форматування* для рядків (працює за аналогією з printf () з Сі), яка використовує той же символ, що і взяття залишку від ділення:

>>> Print («Привіт,% s!»% «світ»)

Привіт, світ!

Python має зручні *ланцюгові порівняння*. Такі умови в програмах — не рідкість:

1 <= a <10 **and** 1 <= b <20

Крім того, логічні операції (or і and) є [ледачими](http://uk.wikipedia.org/w/index.php?title=%D0%9B%D0%B5%D0%B4%D0%B0%D1%87%D1%96_%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F&action=edit&redlink=1): якщо для обчислення значення операції досить першого операнда, цей операнд і є результатом, інакше обчислюється другий операнд логічної операції. Це грунтується на властивостях [алгебри логіки](http://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0_%D0%BB%D0%BE%D0%B3%D1%96%D0%BA%D0%B8): наприклад, якщо один аргумент операції «АБО» (or) є істиною, то і результат цієї операції завжди є істиною. У разі, якщо другий операнд є складним виразом, це дозволяє уникнути витрат на його обчислення. Цей факт широко використовувався до версії 2.5 замість умовної конструкції:

(a < b) **and** "менше" **or** "більше або дорівнює"

Вбудовані типи даних, як правило, мають особливий синтаксис для своїх літералів (записаних у вихідному коді констант):

«Рядок» + 'рядок' "" «теж рядок» „« u»Юнікод-рядок“

True **or** False *# булеві літерали*

3.14 *# число з плаваючою комою*

012 + 0xA *# числа в вісімковій та шістнадцятковій системах числення*

1 + 2j *# ціле число та уявне число*

[1, 2, «a»] *# список*

(1, 2, «a») *# кортеж*

{'A': 1, 'b': 'B'} *# словник*

**lambda** x: x\*\*2 *# неіменована функція*

Для списків (та інших послідовностей) Python пропонує набір операцій над зрізами. Особливістю є індексація, яка може здатися новачкові дивною, але розкриває свою узгодженість по мірі використання. Індекси елементів списку починаються з нуля.

Запис зрізу s [N:M] означає, що у зріз потрапляють всі елементи від N включно до M не включно.

**3.1.3. Імена**

Ім'я (ідентифікатор) може починатися з латинської букви будь-якого регістра або підкреслення, після чого в імені можна використовувати і цифри. Як ім'я не можна використовувати ключові слова (їхній список можна дізнатися за import keyword; print keyword.kwlist) і небажано перевизначати вбудовані імена. Імена, що починаються на підкреслення, мають спеціальне значення.

У кожній точці програми інтерпретатор має доступ до трьох [просторів імен](http://uk.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%96%D1%80_%D1%96%D0%BC%D0%B5%D0%BD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)&action=edit&redlink=1) (тобто відображення імен в об'єкти): *локальному, глобальному та вбудованому.*

*Області видимості* імен можуть бути вкладеними один в одного (всередині визначення функції видно імена з навколишнього блоку коду). На практиці з областями видимості та зв'язуванням імен пов'язано кілька правил «гарного тону», про які можна докладніше дізнатися з документації.

**3.1.4. Рядки документації**

Python пропонує механізм [документування](http://uk.wikipedia.org/w/index.php?title=%D0%A2%D0%B5%D1%85%D0%BD%D1%96%D1%87%D0%BD%D0%B0_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D1%96%D1%8F&action=edit&redlink=1) коду pydoc. На початок кожного модулю, класу, функції вставляється рядок документації — [*docstring*](http://uk.wikipedia.org/w/index.php?title=Docstring&action=edit&redlink=1). Рядки документації залишаються в коді на момент виконання, і в мову вбудований доступ до документації, що використовується сучасними IDE (наприклад, [Eclipse](http://uk.wikipedia.org/wiki/Eclipse)).

В інтерактивному режимі можна отримати допомогу, згенерувати гіпертекстову документацію для цілого модуля або навіть застосувати *doctest* для автоматичного тестування модуля.

**3.1.5. Директиви**

Починаючи з Python 2.3, для використання в тексті програми символів, що не входять до ASCII, необхідно явно вказувати кодування вихідного коду на початку модуля, наприклад:

*# -\*- coding: utf-8 -\*-*

Після цього можна, наприклад, використовувати кирилицю в Unicode-літералах.

**3.1.6. Інтерактивний режим**

Подібно Ліспу та Прологу в режимі відлагодження, інтерпретатор Python має інтерактивний режим роботи, при якому введені з клавіатури оператори відразу ж виконуються, а результат виводиться на екран. Цей режим цікавий не тільки новачкам, але й досвідченим програмістам, які можуть протестувати в інтерактивному режимі будь-яку ділянку коду, перш ніж використовувати його в основній програмі, або просто використовувати як калькулятор з великим набором функцій.

Так виглядає спілкування з Python в інтерактивному режимі:

>>> 2 \*\* 100 # піднесення 2 до 100-го степеня

1267650600228229401496703205376L

>>> from math import \* # імпорт математичних функцій

>>> sin (pi \* 0.5) # обчислення синуса від половини пі

1.0

>>> help (sorted) # допомогу по функції sorted

Help on built-in function sorted in module \_\_builtin\_\_:

sorted (…)

sorted (iterable, cmp=none, key=none, reverse=false) -> new sorted list

В інтерактивному режимі доступний дебагер pdb та система довідки (викликається за help()).

Система допомоги працює для модулів, класів і функцій, тільки якщо ті були забезпечені рядками документації.

Крім вбудованої, існує й покращена інтерактивна оболонка IPython.

**3.1.7. Керування контекстом виконання**

У Python 2.5 з'явилися засоби для керування контекстом виконання блоку коду — оператор with та модуль contextlib.

Оператор може застосовуватися в тих випадках, коли 'до' та 'після' деяких дій повинні обов'язково виконуватися деякі інші дії, незалежно від створених у блоці винятків або операторів return: файли повинні бути закриті, ресурси звільнені, перенаправлення стандартного введення/виведення закінчено тощо Оператор покращує читабельність коду, і отже, допомагає уникати помилок.

**3.1.8.Обробка винятків**

[Обробка винятків](http://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%B2%D0%B8%D0%BD%D1%8F%D1%82%D0%BA%D1%96%D0%B2) підтримується в Python за допомогою операторів try, except, else, finally, raise , що утворюють блок обробки виключення. У загальному випадку блок виглядає наступним чином:

**try**:

*# Тут код, в якому може викнути виключення*

**raise** ExceptionType ("message")

**except** (Тип виключення1, Тип виключення2, …), Змінна:

*# Код в блоці виконується, якщо тип винятку збігається з одним з типів*

*# (Тип виключення1, Тип виключення2, ...) або є спадкоємцем одного*

*# з цих типів.*

*# Отриманий викняток доступний в необов'язковій Змінній.*

**except** (Тип виключення3, Тип виключення4, …), Змінна:

*# Кількість блоків except не обмежено*

**raise** *# Згенерувати викняток "поверх" отриманого; без параметрів — повторно згенерувати отримане*

**except**:

*# Буде виконано за будь-якого викнятку, не обробленого типізованими блоками except*

**else**:

*# Код блоку виконується, якщо не було отримано винятків.*

**finally**:

*# Буде виконано в будь-якому випадку, можливо після відповідного*

*# блоку except або else*

Спільне використання else, except і finally стало можливо тільки починаючи з Python 2.5. Інформація про поточний виняток завжди доступна через sys.exc\_info(). Крім значення виключення, Python також зберігає стан стеку аж до точки збудження винятку — так званий traceback.

На відміну від мов програмування, що компілюються, в Python використання винятку не призводить до значних накладних витрат (а часто навіть дозволяє прискорити виконання програм) і дуже широко використовується. Винятки узгоджуються з філософією Python ( «Помилки ніколи не повинні замовчувати») та є одним із засобів підтримки «[качиної типізації»](http://uk.wikipedia.org/w/index.php?title=%D0%9A%D0%B0%D1%87%D0%B8%D0%BD%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F&action=edit&redlink=1).

Іноді, замість явної обробки винятків, зручніше використовувати блок [with](http://uk.wikipedia.org/wiki/Python#.D0.9A.D0.B5.D1.80.D1.83.D0.B2.D0.B0.D0.BD.D0.BD.D1.8F_.D0.BA.D0.BE.D0.BD.D1.82.D0.B5.D0.BA.D1.81.D1.82.D0.BE.D0.BC_.D0.B2.D0.B8.D0.BA.D0.BE.D0.BD.D0.B0.D0.BD.D0.BD.D1.8F) (доступний, починаючи з Python 2.5).

**3.2.Функціональне програмування**

Python підтримує парадигму [функціонального програмування](http://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), зокрема:

* Функція є об'єктом.
* Функції вищих порядків.
* Рекурсія.
* Розвинена обробка списків (спискові вирази, операції над послідовностями, ітератори).
* Аналог замикань.
* Часткове застосування функції.
* Можливість реалізації інших засобів на самій мові (наприклад, [каррінг](http://uk.wikipedia.org/w/index.php?title=%D0%9A%D0%B0%D1%80%D1%80%D1%96%D0%BD%D0%B3&action=edit&redlink=1)).

Найпростішими є [спискові висловлювання](http://uk.wikipedia.org/w/index.php?title=%D0%A1%D0%BF%D0%B8%D1%81%D0%BA%D0%BE%D0%B2%D1%96_%D0%B2%D0%B8%D1%81%D0%BB%D0%BE%D0%B2%D0%BB%D1%8E%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F&action=edit&redlink=1) (*list comprehension*). Наприклад, щоб отримати список квадратів натуральних чисел, менших 10, можна вжити вислів

l = [x\*\*2 **for** x **in** range(10)]

Підтримуються [анонімні функції](http://uk.wikipedia.org/w/index.php?title=%D0%90%D0%BD%D0%BE%D0%BD%D1%96%D0%BC%D0%BD%D1%96_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%97&action=edit&redlink=1):

add2 = **lambda** x: x+2

після чого add2(5) == 7. Такі конструкції корисні, наприклад, як аргументи у функціональних висловлюваннях: filter(lambda x: x < 5, somelist) вибере зі списку somelist лише значення, менші 5 (тут lambda конструює справді анонімну функцію). Такий самий результат можна одержати за допомогою спискового висловлювання [x for x in somelist if x < 5]. Анонімні функції, як і інші функції, можна створювати всередині інших функцій, реалізуючи повноцінні [замикання](http://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%BC%D0%B8%D0%BA%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)).

Мова має чіткий і послідовний синтаксис, продуману модульність та [масштабованість](http://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D1%96%D1%81%D1%82%D1%8C), завдяки чому вихідний код, написаних на Python програм, легко читається.

**3.3.Об'єктно-орієнтоване програмування**

Дизайн мови Python побудований навколо об'єктно-орієнтованої моделі програмування. Реалізація [ООП](http://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) в Python є елегантною, потужною та добре продуманою, але разом з тим, достатньо специфічною в порівнянні з іншими [об'єктно-орієнтованими мовами](http://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F).

**3.3.1.Можливості та особливості:**

1. Класи є одночасно об'єктами з усіма нижче наведеними можливостями.
2. Успадкування, в тому числі множинне.
3. Поліморфізм (всі функції віртуальні).
4. Інкапсуляція (два рівні — загальнодоступні та приховані методи і поля). Особливість — приховані члени доступні для використання та помічені як приховані лише особливими іменами.
5. Спеціальні методи, керуючі життєвим циклом об'єкта: конструктори, деструктори, розподільники пам'яті.
6. Перевантаження операторів (усіх, крім is, '.', '=' і символьних логічних).
7. Властивості (імітація поля за допомогою функцій).
8. Управління доступу до полів (емуляція полів і методів, частковий доступ, тощо).
9. Методи для управління найбільш поширеними операціями (істинносне значення, len(), глибоке копіювання, [серіалізація](http://uk.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D1%96%D0%B0%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F), ітерація по об'єкту, …)
10. Метапрограмування (управління створенням класів, тригери на створення класів, та ін.)
11. Повна [інтроспекція](http://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D1%80%D0%BE%D1%81%D0%BF%D0%B5%D0%BA%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) (можливість визначати тип і структуру об’єкта під час виконання програми, може використовуватися для реалізації поліморфізму).
12. Класові та статичні методи, класові поля.
13. Класи, вкладені у функції та інші класи.

**3.3.2.Модулі та пакети**

[Програмне забезпечення](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) ([застосунок](http://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) або [бібліотека](http://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%B1%D0%BB%D1%96%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))) на Python оформлюється у вигляді [модулів](http://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), які у свою чергу можуть бути зібрані в *пакунки*. Модулі можуть розташовуватися як у [каталогах](http://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3_(%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)), так і в [ZIP-архівах](http://uk.wikipedia.org/wiki/ZIP). Модулі можуть бути двох типів за своїм походженням: модулі, написані на «чистому» Python, і модулі розширення (extension modules), написані на інших мовах програмування.

Наприклад, в стандартній бібліотеці є «чистий» модуль *pickle* і його аналог на Сі: *cPickle*. Модуль оформляється у вигляді окремого файлу, а пакет — у вигляді окремого каталогу. Підключення модулю до програми здійснюється оператором **import**. Після імпорту модуль представлений окремим об'єктом, що дає доступ до простору імен модуля. У ході виконання програми модуль можна перезавантажити функцією reload().

**3.3.3.Інтроспекція**

Python підтримує повну [інтроспекцію](http://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D1%80%D0%BE%D1%81%D0%BF%D0%B5%D0%BA%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) часу виконання. Це означає, що для будь-якого об'єкта можна отримати всю інформацію про його внутрішню структуру.

Застосування інтроспекції ([метапрограмування](http://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B0%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) є важливою частиною того, що називають «pythonic style», і широко застосовується в бібліотеках і [фреймворках](http://uk.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) Python, таких як [PyRO](http://uk.wikipedia.org/w/index.php?title=PyRO&action=edit&redlink=1), [PLY](http://uk.wikipedia.org/w/index.php?title=PLY&action=edit&redlink=1), [Cherry](http://uk.wikipedia.org/w/index.php?title=Cherry&action=edit&redlink=1), [Django](http://uk.wikipedia.org/wiki/Django) та інших, заощаджуючи час програміста, що ними користується.

**3.3.4.Ітератори**

У програмах на Python широко використовуються [ітератори](http://uk.wikipedia.org/wiki/%D0%86%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80). Цикл for може працювати як з послідовністю, так і з ітераторами. Усі колекції, як правило, надають ітератор. Об'єкти визначеного користувачем класу теж можуть бути ітераторами.

Модуль itertools стандартної бібліотеки містить багато корисних функцій для роботи з ітераторами.

**3.3.5.Генератори**

Однією з цікавих можливостей мови є **генератори** — функції, що зберігають внутрішній стан: значення локальних змінних і поточну інструкцію. Генератори можуть використовуватися як [ітератор](http://uk.wikipedia.org/wiki/%D0%86%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80) і для структур даних і для [ледачих обчислень](http://uk.wikipedia.org/w/index.php?title=%D0%9B%D0%B5%D0%B4%D0%B0%D1%87%D1%96_%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F&action=edit&redlink=1).

При виклику генератора функція негайно повертає об'єкт-ітератор, який зберігає поточну точку виконання та стан локальних змінних функції. При запиті наступного значення (за допомогою методу next(), неявно викликається в for циклі) генератор продовжує виконання функції від попередньої точки зупину до наступного оператора yield або return.

У Python 2.4 з'явилися **генераторні вирази** — вирази, що дають у результаті генератор. Генераторні вирази дозволяють заощадити пам'ять там, де інакше потрібно було б використовувати список із проміжними результатами:

>>> sum(i **for** i **in** xrange (1, 100) **if** i % 2 != 0)

2500

У цьому прикладі підсумовуються всі непарні числа від 1 до 99.

Починаючи з версії 2.5, Python підтримує повноцінні *співпроцедури:* тепер в генератор можна передавати значення за допомогою методу send() та збуджувати в його контексті виняток за допомогою методу throw().

**3.3.6.Декоратори**

Починаючи з версії 2.4, Python дозволяє використовувати, так звані, *Декоратори* (не слід плутати з однойменним [шаблоном проектування](http://uk.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%B8_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) для підтримки існуючої практики перетворення функцій та методів у місці визначення (декораторів може бути декілька). Після довгих дебатів для декораторів став використовуватися символ @ у рядках, що передують визначенню функції або методу. Наступний приклад містить опис статичного методу без застосування декоратора:

**def** myWonderfulMethod ():

**return** "Деякий метод"

myWonderfulMethod = staticmethod (myWonderfulMethod)

і за допомогою декоратора:

@staticmethod

**def** myWonderfulMethod ():

**return** "Деякий метод"

Декоратор є нічим іншим, як функцією, що одержує в якості першого аргументу функцію, що декорується або метод. Декоратори можна вважати елементом [*аспектно-орієнтованого програмування*](http://uk.wikipedia.org/wiki/%D0%90%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)*.*

З версії 2.6 декоратори можна використовувати з класами, аналогічно функціям.

**3.3.7.Інші можливості**

У Python є ще кілька можливостей, що відрізняють його від багатьох інших мов високою гнучкістю та динамічністю.

Наприклад, клас є об'єктом, а в операторі визначення класу можна використовувати вирази в списку батьківських класів.

**def** getClass():

**return** dict

**class** D(getClass()):

**pass**

d = D()

Можна модифікувати багато об'єктів під час виконання, наприклад класи:

>>> **class** X(object): **pass**

…

>>> y = X()

>>> y.wrongMethod() *# такого методу поки немає*

Traceback (most recent call last):

File "<stdin>", line 1, **in** <module>

AttributeError: 'X' object has no attribute 'wrongMethod'

>>> X.wrongMethod = **lambda** self : 'im here' *# додамо його*

>>> y.wrongMethod() *# так як доступ до методу призводить до пошуку по \_\_dict\_\_ класу,*

'im here' *# то wrongMethod стає доступним всім екземплярам*

**4. Google Docs**

**4.1. Google Docs**

**Google Docs** (укр. *Документи Ґуґл*) — розроблений Google безкоштовний мережевий офісний пакет, що включає текстовий, табличний редактор і службу для створення презентацій.

Утворений у результаті злиття Writely і Google Spreadsheets.

Це веб-орієнтована програма, що працює в рамках веб-браузера без установлення на комп'ютер користувача. Документи і таблиці, що створюються користувачем, зберігаються на сервері Google, або можуть бути збережені у файл. Це одна з ключових переваг програми, оскільки доступ до введених даних може здійснюватися з будь-якого комп’ютера, під’єднаного до інтернету. Доступ до особистих документів захищений паролем.

**4.1.**[**1 Writely**](http://uk.wikipedia.org/wiki/Google_Docs#Writely)

Це текстовий процесор, що дозволяє редагувати текстові документи OpenDocument, Microsoft Word, а також електронні таблиці. Створений за допомогою технології AJAX.

Доступна велика кількість засобів форматування: зміна розміру і стилю шрифту, вибір кольору та оздоблення, створення списків і таблиць, вставка картинок, посилань і спеціальних знаків. Можна робити закладки, коментарі.

Зберігаються документи автоматично, з мірою внесення змін, але кожна правка записується, і можна користуватися функцією скасування і повернення змін так само, як і у звичайному текстовому редакторі. Існує можливість завантажувати на сервер і скачувати з нього файли у різних форматах.

Підтримується простий текст, HTML, Microsoft Word, RTF, OpenDocument, PDF і кілька графічних форматів. Можна отримати підбірку текстів у вигляді файлів HTML в архіві ZIP. Заявлена (але поки не реалізована) підтримка Word Perfect.

**4.1.2.** [**Google Spreadsheets**](http://uk.wikipedia.org/wiki/Google_Docs#Google_Spreadsheets)

Дозволяє заносити дані в рядки і стовпчики електронної таблиці, а також проводити нескладні обчислення. Програма допускає перетворення даних і у формат Microsoft Excel, і в загальнодоступні OpenDocument і CSV.

Передбачена можливість доступу інших користувачів до створеної таблиці, як з правом редагування, так і без.

Це програмне забезпечення має трошки менше можливостей, порівняно з повноцінними електронними таблицями, такими, як Microsoft Excel або OpenOffice.org Calc , але цілком достатньо для простих дій з даними і воно постійно вдосконалюється, набуваючи все нових і нових функцій.

**4.1.3 Google Presentations**

Дає можливість створювати електронні презентації. Реалізована можливість імпорту/експорту файлів Microsoft PowerPoint (\*.ppt).

У основу Google Presentations лягли розробки компанії Tonic System, придбаної компанією Google в 2007 році.

**4.1.4.Історія**

Google Docs & Spreadsheets заснований на двох різних продуктах, *Writely* і *Google Spreadsheets*. Writely був окремим текстовим процесором для інтернету. Він був створений компанією Upstartle, і включав такі функції, як спільне редагування і контроль доступу. Меню, гарячі клавіші і діалогові вікна повторювали ті, що користувачі могли бачити в текстових процесорах, встановлених у них на комп'ютері, таких як Microsoft Word або OpenOffice.org Writer.

На початку 2006 року Google придбала Upstartle. На той момент в Upstartle працювало четверо співробітників. Writely закрив реєстрації на своєму сервісі на час перенесення на сервера Google. У серпні 2006 року Writely почав розсилати запрошення тим, хто залишив запит на отримання інформації про початок роботи до кінця літа. 23 серпня 2006 року Writely розіслав запрошення всім, хто залишав запит, і став доступним для всіх. Клієнти Writely могли запросити інших, додаючи їх в співавторів свого документа на Writely. Writely продовжував працювати з своєю системою користувачів до 19 вересня 2006, поки не був включений в систему Google Accounts.

Writely спочатку працював на технології Microsoft ASP.NET, яка використовує Microsoft Windows. З липня 2006 сервери Writely працюють на операційних системах, заснованих на Linux.

Одночасно з цим Google розробляла Google Spreadsheets. Google представила Spreadsheets 6 червня 2006, і зробила його доступним тільки для обмеженого числа користувачів, що реєструються першими. Тестова версія для обмеженого числа користувачів була завершена, і вона була замінена бета-версією, доступною для всіх власників облікових записів Google Account.

**5. Практична частина**

**5.1. Інтерпретація результатів.**

Для демонстрації можливостей мови Python я вирішила написати програму, за допомогою якої можна відправити певні дані в Google Spread Sheets не вводячи їх у безпосередньо у таблицю, а використовуючи в зручне віконечко відправляти одним натиском мишки на відповідній кнопці цілу групу інформації.

Для цього я заздалегідь створила документ і Google Spread Sheets ,і контролюю його редагування як власник ,але переглядати та редагувати таблицю можуть усі, хто отримає на неї посилання.

Документи в Google Docs можуть лише власники акантів у Google.

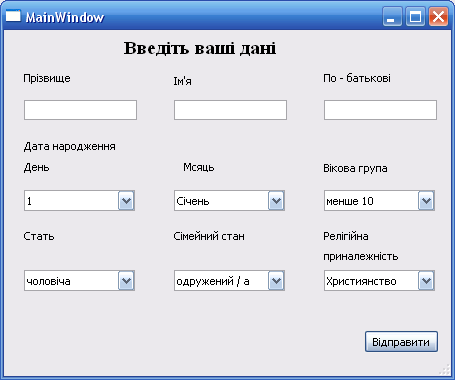
**5.2. Розробка програми**

**5.2.1. Проектування інтерфейсу.**

Для створення графічного інтерфейсу я використала бібліотеки Ot, а саме обрала PyQt4. Оболонка, як на мене, має дуже зручний інтерфейс і багато корисних та цікавих можливостей.

Використання такого середовище дозволяє заощадити багато часу, який зайняло б програмування інтерфейсу вручну.

Отож в Qt Designer я створила таке вікно :



**5.2.2. Компілювання програми.**

Щоб цей інтерфейс можна було використовувати в моїй програмі його слід конвертувати у файл з розширенням \*.py .

Для цього ми відкриваємо командний рядок і пишемо там такий текст :

**pyuic4 form1.ui > editor.py**

де : **pyuic4** глобальна змінна системи, замість неї можна прописати повний шлях до цього файлу; **form1.ui** - ім’я файлу, що конвертується, а – ім’я файлу, в який конвертується.

Тоді в середовищі Python IDLE написала код програми, використовуючи спеціальні бібліотеки для доступу до Spread Sheets на мові Python, що містяться в додатковому модулі gdata-2.0.15, який можна безкоштовно завантажити з сайту Google.

Файл коду має розширення \*.py.

Проте ,щоб скомпілювати програму і створити виконуваний файл потрібно проробити ще кілька операцій в командному рядку. Складна компільованість вважається головним недоліком цієї мови.

Отож, для створення \*.exe спочатку потрібно створити \*.spec . Для цього пишемо в командній стрічці наступний код :

<шлях до програми >\pyinstaller-1.5.1 >python Makespec.py --onefile -w <шлях до скрипту >\\*.py

В результаті успішного виконання цієї команди утвориться файл \*.spec і папка для проекту всередині директорії <шлях до програми >\pyinstaller-1.5.1.

Далі пишемо наступну команду для створення безпосередньо файлу \*.exe :

<шлях до програми >\pyinstaller-1.5.1 >python Build.py \*\\*.spec

В останніх двох командах python - глобальна змінна ,вказівник на файл python.exe, замість цього можна вказати повний шлях до файлу.

Запустивши отриманий файл ми побачиво вікно, зображене вище. Введемо у відповідні комірки прізвище ім’я та по-батькові, виберемо дату народження, вік стать, сімейний стан та релігійну приналежність. Далі натискаємо кнопку «Відправити».

Перейшовши за наступним посиланням ми побачимо документ Google Spread Sheets – табличку ,в якій відображені введені нами дані.

<https://docs.google.com/spreadsheet/ccc?key=0AsFKUoTWv9sgdC01dFR1WVZ4RjVlc2JFbmY5VXVIWEE>

Для коректної роботи програми необхідно постійне підключення до інтернету, оскільки її робота базується на з’єднанні з серверами Google.

**Висновок**

**Пайтон (Python)** - це потужна мова програмування, якою легко оволодіти. Вона має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування.

Елегантний синтаксис Пайтона, динамічна обробка типів, а також те, що це інтерпретована мова, роблять його ідеальним для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.

Реалізувати принципи ООП в цій мові дуже легко та зручно. Є широкий асортимент безкоштовних середовищ, модулів, бібліотек та графічних оболонок для роботи з Пайтоном, тому кожен користувач неодмінно знайде ідеальну для себе.

Інтерпретатор мови Пайтон і багата стандартна бібліотека (як код-джерело, так і бінарні дистрибутиви для усіх головних операційних систем) можуть бути отримані з сайту Пайтона, і можуть вільно розповсюджуватися. Цей же сайт має дистрибутиви та посилання на численні модулі, програми, утиліти та додаткову документацію.

Завдяки портованості код на Пайтоні легко поєднувати з програмами, які написані іншими мовами.

Інтерпретатор мови Пайтон може бути легко розширений функціями та типами даних, розробленими на C чи C++ (або на іншій мові, яку можна викликати із C). Пайтон також зручний як мова розширення для прикладних програм, що потребують подальшого налагодження.

**Література**

1. Guido van Rossum, Fred L. Drake, Jr.

«Python Tutorial»

1. http://www.python.org/
2. Хвищун І. О. Програмування і математичне моделювання : Підруч. – К.: Видавничий дім «Ін Юре», 2007. – 544с.
3. Абакумова О. О., Козачук О. С.

**«Основні поняття об’єктно-орієнтованого програмування»**

1. Saikon Blog /Development Blog http://saikon.org.ua/?p=21
2. http://younglinux.info/oopython.php
3. http://code.google.com/intl/uk-UA/apis/spreadsheets/data/1.0/developers\_guide\_python.html
4. http://uk.wikipedia.org/
5. http://manuals.u-medio.com/index.php?title=%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%B8\_Python\_%D0%B2\_%D0%BA%D0%BE%D1%80%D0%BE%D1%82%D0%BA%D0%BE%D0%BC%D1%83\_%D0%B2%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D1%96.\_%D0%93%D0%BB%D0%B0%D0%B2%D0%B0\_1

**Додатки**

**Додаток 1. Код програми**

# code.py

# Virusya Lytvyn

# -\*- coding: utf-8 -\*-

import sys

import gdata.spreadsheet.service

import gdata.service

import atom.service

import gdata.spreadsheet

import atom

import gdata

from PyQt4 import QtCore, QtGui

from editor import Ui\_MainWindow

class GoogleSpreadSheet(QtGui.QMainWindow):

def \_\_init\_\_(self, parent=None):

QtGui.QWidget.\_\_init\_\_(self, parent)

self.ui = Ui\_MainWindow()

self.ui.setupUi(self)

self.connect(self.ui.sendButton, QtCore.SIGNAL("clicked()"), self.readDataFromForm)

def readDataFromForm(self):

#print "button %s was pressed" % self.sender()

firstName = unicode(self.ui.firstnameLineEdit.text())

secondName = unicode(self.ui.secondnameLineEdit.text())

lastName = unicode(self.ui.lastnameLineEdit.text())

birthday = unicode(self.ui.CBday.itemText(self.ui.CBday.currentIndex()))

month = unicode(self.ui.CBmounce.itemText(self.ui.CBmounce.currentIndex()))

year = unicode(self.ui.CByear.itemText(self.ui.CByear.currentIndex()))

malefemale = unicode(self.ui.CBmale.itemText(self.ui.CBmale.currentIndex()))

married = unicode(self.ui.CBmerr.itemText(self.ui.CBmerr.currentIndex()))

religious = unicode(self.ui.CBChist.itemText(self.ui.CBChist.currentIndex()))

## firstName = unicode(firstName)

## secondName = unicode(secondName)

## print married

## print religious

gd\_client = gdata.spreadsheet.service.SpreadsheetsService()

gd\_client.email = 'baterfluy@gmail.com'

gd\_client.password = '369741369'

gd\_client.ProgrammaticLogin()

spreadsheet\_feed = gd\_client.GetSpreadsheetsFeed()

for spreadsheet in spreadsheet\_feed.entry:

print spreadsheet.id.text.rsplit('/', 1)[1]

spreadsheet\_key = spreadsheet.id.text.rsplit('/', 1)[1]

feed = gd\_client.GetWorksheetsFeed(spreadsheet\_key)

for worksheet in feed.entry:

worksheet\_key = worksheet.id.text.rsplit('/', 1)[1]

print worksheet\_key

gd\_client = gdata.spreadsheet.service.SpreadsheetsService(spreadsheet\_key, worksheet\_key)

gd\_client.email = 'baterfluy@gmail.com'

gd\_client.password = '369741369'

gd\_client.ProgrammaticLogin()

dict = {'firstname':firstName, 'secondname':secondName, 'lastname':lastName,

'birthdayday':birthday, 'birthdaymonth':month, 'age':year,

'malefemale':malefemale, 'marriage':married, 'religion':religious}

#dict = {'month':month}

#gd\_client.InsertRow(dict, spreadsheet\_key, worksheet\_key)

entry = gd\_client.InsertRow(dict, spreadsheet\_key, worksheet\_key)

if isinstance(entry, gdata.spreadsheet.SpreadsheetsList):

print "Insert row is success."

#Run the program

if \_\_name\_\_ == "\_\_main\_\_":

app = QtGui.QApplication(sys.argv)

myapp = GoogleSpreadSheet()

myapp.show()

sys.exit(app.exec\_())

**Додаток 2. Код інтерфейсу**

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'form.ui'

#

# Created: Sun Dec 04 21:34:17 2011

# by: PyQt4 UI code generator 4.8.6

#

# WARNING! All changes made in this file will be lost!

from PyQt4 import QtCore, QtGui

try:

\_fromUtf8 = QtCore.QString.fromUtf8

except AttributeError:

\_fromUtf8 = lambda s: s

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName(\_fromUtf8("MainWindow"))

MainWindow.resize(447, 346)

MainWindow.setWindowTitle(QtGui.QApplication.translate("MainWindow", "MainWindow", None, QtGui.QApplication.UnicodeUTF8))

self.centralwidget = QtGui.QWidget(MainWindow)

self.centralwidget.setObjectName(\_fromUtf8("centralwidget"))

self.firstnameLineEdit = QtGui.QLineEdit(self.centralwidget)

self.firstnameLineEdit.setGeometry(QtCore.QRect(20, 70, 113, 20))

self.firstnameLineEdit.setObjectName(\_fromUtf8("firstnameLineEdit"))

self.label = QtGui.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(120, 10, 161, 16))

font = QtGui.QFont()

font.setFamily(\_fromUtf8("Times New Roman"))

font.setPointSize(14)

font.setBold(True)

font.setWeight(75)

self.label.setFont(font)

self.label.setLocale(QtCore.QLocale(QtCore.QLocale.Ukrainian, QtCore.QLocale.Ukraine))

self.label.setLineWidth(5)

self.label.setText(QtGui.QApplication.translate("MainWindow", "Р’РІРµРґС–С‚СЊ РІР°С€С– РґР°РЅС– ", None, QtGui.QApplication.UnicodeUTF8))

self.label.setObjectName(\_fromUtf8("label"))

self.label\_2 = QtGui.QLabel(self.centralwidget)

self.label\_2.setGeometry(QtCore.QRect(20, 40, 111, 16))

self.label\_2.setText(QtGui.QApplication.translate("MainWindow", "РџСЂС–Р·РІРёС‰Рµ ", None, QtGui.QApplication.UnicodeUTF8))

self.label\_2.setObjectName(\_fromUtf8("label\_2"))

self.label\_3 = QtGui.QLabel(self.centralwidget)

self.label\_3.setGeometry(QtCore.QRect(170, 40, 91, 21))

self.label\_3.setText(QtGui.QApplication.translate("MainWindow", "Р†Рј\'СЏ ", None, QtGui.QApplication.UnicodeUTF8))

self.label\_3.setObjectName(\_fromUtf8("label\_3"))

self.label\_4 = QtGui.QLabel(self.centralwidget)

self.label\_4.setGeometry(QtCore.QRect(320, 40, 81, 16))

self.label\_4.setText(QtGui.QApplication.translate("MainWindow", "РџРѕ - Р±Р°С‚СЊРєРѕРІС– ", None, QtGui.QApplication.UnicodeUTF8))

self.label\_4.setObjectName(\_fromUtf8("label\_4"))

self.label\_5 = QtGui.QLabel(self.centralwidget)

self.label\_5.setGeometry(QtCore.QRect(20, 190, 81, 31))

self.label\_5.setText(QtGui.QApplication.translate("MainWindow", "РЎС‚Р°С‚СЊ ", None, QtGui.QApplication.UnicodeUTF8))

self.label\_5.setObjectName(\_fromUtf8("label\_5"))

self.label\_6 = QtGui.QLabel(self.centralwidget)

self.label\_6.setGeometry(QtCore.QRect(170, 190, 101, 31))

self.label\_6.setText(QtGui.QApplication.translate("MainWindow", "РЎС–РјРµР№РЅРёР№ СЃС‚Р°РЅ ", None, QtGui.QApplication.UnicodeUTF8))

self.label\_6.setObjectName(\_fromUtf8("label\_6"))

self.label\_7 = QtGui.QLabel(self.centralwidget)

self.label\_7.setGeometry(QtCore.QRect(320, 190, 111, 31))

self.label\_7.setText(QtGui.QApplication.translate("MainWindow", "Р РµР»С–РіС–Р№РЅР°", None, QtGui.QApplication.UnicodeUTF8))

self.label\_7.setObjectName(\_fromUtf8("label\_7"))

self.CBday = QtGui.QComboBox(self.centralwidget)

self.CBday.setGeometry(QtCore.QRect(20, 160, 111, 21))

self.CBday.setFrame(True)

self.CBday.setModelColumn(0)

self.CBday.setObjectName(\_fromUtf8("CBday"))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(0, QtGui.QApplication.translate("MainWindow", "1", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(1, QtGui.QApplication.translate("MainWindow", "2", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(2, QtGui.QApplication.translate("MainWindow", "3", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(3, QtGui.QApplication.translate("MainWindow", "4", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(4, QtGui.QApplication.translate("MainWindow", "5", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(5, QtGui.QApplication.translate("MainWindow", "6", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(6, QtGui.QApplication.translate("MainWindow", "7", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(7, QtGui.QApplication.translate("MainWindow", "8", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(8, QtGui.QApplication.translate("MainWindow", "9", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(9, QtGui.QApplication.translate("MainWindow", "10", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(10, QtGui.QApplication.translate("MainWindow", "11", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(11, QtGui.QApplication.translate("MainWindow", "12", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(12, QtGui.QApplication.translate("MainWindow", "13", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(13, QtGui.QApplication.translate("MainWindow", "14", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(14, QtGui.QApplication.translate("MainWindow", "15", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(15, QtGui.QApplication.translate("MainWindow", "16", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(16, QtGui.QApplication.translate("MainWindow", "17", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(17, QtGui.QApplication.translate("MainWindow", "18", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(18, QtGui.QApplication.translate("MainWindow", "19", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(19, QtGui.QApplication.translate("MainWindow", "20", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(20, QtGui.QApplication.translate("MainWindow", "21", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(21, QtGui.QApplication.translate("MainWindow", "22", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(22, QtGui.QApplication.translate("MainWindow", "23", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(23, QtGui.QApplication.translate("MainWindow", "24", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(24, QtGui.QApplication.translate("MainWindow", "25", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(25, QtGui.QApplication.translate("MainWindow", "26", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(26, QtGui.QApplication.translate("MainWindow", "27", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(27, QtGui.QApplication.translate("MainWindow", "28", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(28, QtGui.QApplication.translate("MainWindow", "29", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(29, QtGui.QApplication.translate("MainWindow", "30", None, QtGui.QApplication.UnicodeUTF8))

self.CBday.addItem(\_fromUtf8(""))

self.CBday.setItemText(30, QtGui.QApplication.translate("MainWindow", "31", None, QtGui.QApplication.UnicodeUTF8))

self.CBmerr = QtGui.QComboBox(self.centralwidget)

self.CBmerr.setGeometry(QtCore.QRect(170, 240, 111, 21))

self.CBmerr.setObjectName(\_fromUtf8("CBmerr"))

self.CBmerr.addItem(\_fromUtf8(""))

self.CBmerr.setItemText(0, QtGui.QApplication.translate("MainWindow", "РѕРґСЂСѓР¶РµРЅРёР№ / Р°", None, QtGui.QApplication.UnicodeUTF8))

self.CBmerr.addItem(\_fromUtf8(""))

self.CBmerr.setItemText(1, QtGui.QApplication.translate("MainWindow", "РЅРµРѕРґСЂСѓР¶РµРЅРёР№ / Р°", None, QtGui.QApplication.UnicodeUTF8))

self.CBmerr.addItem(\_fromUtf8(""))

self.CBmerr.setItemText(2, QtGui.QApplication.translate("MainWindow", "Р±Р°Р№РґСѓР¶РёР№ / Р°", None, QtGui.QApplication.UnicodeUTF8))

self.CBChist = QtGui.QComboBox(self.centralwidget)

self.CBChist.setGeometry(QtCore.QRect(320, 240, 111, 21))

self.CBChist.setObjectName(\_fromUtf8("CBChist"))

self.CBChist.addItem(\_fromUtf8(""))

self.CBChist.setItemText(0, QtGui.QApplication.translate("MainWindow", "РҐСЂРёСЃС‚РёСЏРЅСЃС‚РІРѕ", None, QtGui.QApplication.UnicodeUTF8))

self.CBChist.addItem(\_fromUtf8(""))

self.CBChist.setItemText(1, QtGui.QApplication.translate("MainWindow", "Р†СЃР»Р°Рј", None, QtGui.QApplication.UnicodeUTF8))

self.CBChist.addItem(\_fromUtf8(""))

self.CBChist.setItemText(2, QtGui.QApplication.translate("MainWindow", "РџСЂРѕС‚РёСЃС‚Р°РЅСЃС‚РІРѕ", None, QtGui.QApplication.UnicodeUTF8))

self.CBChist.addItem(\_fromUtf8(""))

self.CBChist.setItemText(3, QtGui.QApplication.translate("MainWindow", "Р†СѓРґР°С—Р·Рј", None, QtGui.QApplication.UnicodeUTF8))

self.CBChist.addItem(\_fromUtf8(""))

self.CBChist.setItemText(4, QtGui.QApplication.translate("MainWindow", "Р‘СѓРґРґРёР·Рј ", None, QtGui.QApplication.UnicodeUTF8))

self.CBChist.addItem(\_fromUtf8(""))

self.CBChist.setItemText(5, QtGui.QApplication.translate("MainWindow", "РљРѕРЅС„СѓС†С–Р°РЅСЃС‚РІРѕ", None, QtGui.QApplication.UnicodeUTF8))

self.CBChist.addItem(\_fromUtf8(""))

self.CBChist.setItemText(6, QtGui.QApplication.translate("MainWindow", "С–РЅС€Р°", None, QtGui.QApplication.UnicodeUTF8))

self.label\_8 = QtGui.QLabel(self.centralwidget)

self.label\_8.setGeometry(QtCore.QRect(320, 210, 111, 31))

self.label\_8.setText(QtGui.QApplication.translate("MainWindow", "РїСЂРёРЅР°Р»РµР¶РЅС–СЃС‚СЊ", None, QtGui.QApplication.UnicodeUTF8))

self.label\_8.setObjectName(\_fromUtf8("label\_8"))

self.label\_9 = QtGui.QLabel(self.centralwidget)

self.label\_9.setGeometry(QtCore.QRect(20, 130, 46, 13))

self.label\_9.setText(QtGui.QApplication.translate("MainWindow", "Р”РµРЅСЊ ", None, QtGui.QApplication.UnicodeUTF8))

self.label\_9.setObjectName(\_fromUtf8("label\_9"))

self.label\_10 = QtGui.QLabel(self.centralwidget)

self.label\_10.setGeometry(QtCore.QRect(180, 130, 46, 13))

self.label\_10.setText(QtGui.QApplication.translate("MainWindow", "РњСЃСЏС†СЊ ", None, QtGui.QApplication.UnicodeUTF8))

self.label\_10.setObjectName(\_fromUtf8("label\_10"))

self.label\_11 = QtGui.QLabel(self.centralwidget)

self.label\_11.setGeometry(QtCore.QRect(320, 130, 71, 16))

self.label\_11.setText(QtGui.QApplication.translate("MainWindow", "Р’С–РєРѕРІР° РіСЂСѓРїР°", None, QtGui.QApplication.UnicodeUTF8))

self.label\_11.setObjectName(\_fromUtf8("label\_11"))

self.label\_12 = QtGui.QLabel(self.centralwidget)

self.label\_12.setGeometry(QtCore.QRect(20, 100, 131, 31))

self.label\_12.setText(QtGui.QApplication.translate("MainWindow", "Р”Р°С‚Р° РЅР°СЂРѕРґР¶РµРЅРЅСЏ ", None, QtGui.QApplication.UnicodeUTF8))

self.label\_12.setObjectName(\_fromUtf8("label\_12"))

self.CBmale = QtGui.QComboBox(self.centralwidget)

self.CBmale.setGeometry(QtCore.QRect(20, 240, 111, 21))

self.CBmale.setObjectName(\_fromUtf8("CBmale"))

self.CBmale.addItem(\_fromUtf8(""))

self.CBmale.setItemText(0, QtGui.QApplication.translate("MainWindow", "С‡РѕР»РѕРІС–С‡Р°", None, QtGui.QApplication.UnicodeUTF8))

self.CBmale.addItem(\_fromUtf8(""))

self.CBmale.setItemText(1, QtGui.QApplication.translate("MainWindow", "Р¶С–РЅРѕС‡Р°", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce = QtGui.QComboBox(self.centralwidget)

self.CBmounce.setGeometry(QtCore.QRect(170, 160, 111, 21))

self.CBmounce.setObjectName(\_fromUtf8("CBmounce"))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(0, QtGui.QApplication.translate("MainWindow", "РЎС–С‡РµРЅСЊ", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(1, QtGui.QApplication.translate("MainWindow", "Р›СЋС‚РёР№", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(2, QtGui.QApplication.translate("MainWindow", "Р‘РµСЂРµР·РµРЅСЊ", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(3, QtGui.QApplication.translate("MainWindow", "РљРІС–С‚РµРЅСЊ", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(4, QtGui.QApplication.translate("MainWindow", "РўСЂР°РІРµРЅСЊ", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(5, QtGui.QApplication.translate("MainWindow", "Р§РµСЂРІРµРЅСЊ", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(6, QtGui.QApplication.translate("MainWindow", "Р›РёРїРµРЅСЊ", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(7, QtGui.QApplication.translate("MainWindow", "РЎРµСЂРїРµРЅСЊ", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(8, QtGui.QApplication.translate("MainWindow", "Р’РµСЂРµСЃРµРЅСЊ", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(9, QtGui.QApplication.translate("MainWindow", "Р–РѕРІС‚РµРЅСЊ", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(10, QtGui.QApplication.translate("MainWindow", "Р›РёСЃС‚РѕРїР°Рґ", None, QtGui.QApplication.UnicodeUTF8))

self.CBmounce.addItem(\_fromUtf8(""))

self.CBmounce.setItemText(11, QtGui.QApplication.translate("MainWindow", "Р“СЂСѓРґРµРЅСЊ", None, QtGui.QApplication.UnicodeUTF8))

self.CByear = QtGui.QComboBox(self.centralwidget)

self.CByear.setGeometry(QtCore.QRect(320, 160, 111, 21))

self.CByear.setObjectName(\_fromUtf8("CByear"))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(0, QtGui.QApplication.translate("MainWindow", "РјРµРЅС€Рµ 10", None, QtGui.QApplication.UnicodeUTF8))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(1, QtGui.QApplication.translate("MainWindow", "10 - 15", None, QtGui.QApplication.UnicodeUTF8))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(2, QtGui.QApplication.translate("MainWindow", "15 - 20", None, QtGui.QApplication.UnicodeUTF8))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(3, QtGui.QApplication.translate("MainWindow", "20 - 30", None, QtGui.QApplication.UnicodeUTF8))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(4, QtGui.QApplication.translate("MainWindow", "30 - 40", None, QtGui.QApplication.UnicodeUTF8))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(5, QtGui.QApplication.translate("MainWindow", "40 - 50", None, QtGui.QApplication.UnicodeUTF8))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(6, QtGui.QApplication.translate("MainWindow", "50 - 55", None, QtGui.QApplication.UnicodeUTF8))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(7, QtGui.QApplication.translate("MainWindow", "55 - 65", None, QtGui.QApplication.UnicodeUTF8))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(8, QtGui.QApplication.translate("MainWindow", "65 - 75", None, QtGui.QApplication.UnicodeUTF8))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(9, QtGui.QApplication.translate("MainWindow", "75 - 80", None, QtGui.QApplication.UnicodeUTF8))

self.CByear.addItem(\_fromUtf8(""))

self.CByear.setItemText(10, QtGui.QApplication.translate("MainWindow", "Р±С–Р»СЊС€Рµ 80", None, QtGui.QApplication.UnicodeUTF8))

self.secondnameLineEdit = QtGui.QLineEdit(self.centralwidget)

self.secondnameLineEdit.setGeometry(QtCore.QRect(170, 70, 113, 20))

self.secondnameLineEdit.setObjectName(\_fromUtf8("secondnameLineEdit"))

self.lastnameLineEdit = QtGui.QLineEdit(self.centralwidget)

self.lastnameLineEdit.setGeometry(QtCore.QRect(320, 70, 113, 20))

self.lastnameLineEdit.setObjectName(\_fromUtf8("lastnameLineEdit"))

self.sendButton = QtGui.QPushButton(self.centralwidget)

self.sendButton.setGeometry(QtCore.QRect(360, 300, 75, 23))

self.sendButton.setText(QtGui.QApplication.translate("MainWindow", "Р’С–РґРїСЂР°РІРёС‚Рё", None, QtGui.QApplication.UnicodeUTF8))

self.sendButton.setObjectName(\_fromUtf8("sendButton"))

MainWindow.setCentralWidget(self.centralwidget)

self.statusbar = QtGui.QStatusBar(MainWindow)

self.statusbar.setObjectName(\_fromUtf8("statusbar"))

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

pass