

РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТІВ МОВОЮ PYTHON ТА ЇХ ВИКОРИСТАННЯ В IPYTHON NOTEBOOK

Показані принципи розробки бібліотеки програмних компонентів мовою Python та її використання в інтерактивному обчислювальному середовищі IPython Notebook. Компоненти реалізуються Python-класами, є незалежними та володіють високим рівнем інтеграції даних, алгоритмів, інтерфейсів користувача і автодокументації з розширеним форматуванням.

Ключові слова: програмний компонент, Python, IPython Notebook, інтерактивне обчислювальне середовище, об'єктно-орієнтоване програмування, клас

Python - це популярна мова програмування загального призначення, яка орієнтована на підвищення продуктивності розробки програмного забезпечення [1]. Python підтримує кілька парадигм програмування (в тому числі об'єктно-орієнтовану), працює майже на усіх відомих платформах, є відкритою і вільною, виконується шляхом інтерпретації байт-коду, код програм компактний і легко читається, мові характерні динамічна типізація, повна інтроспекція, зручні структури даних, велика стандартна бібліотека та значна кількість сторонніх бібліотек різноманітного призначення. IPython Notebook - це інтерактивне обчислювальне середовище, яке ґрунтується на веб та дозволяє користувачам створювати документи, що містять "живий" програмний код, текст, формули LaTeX, HTML, рисунки і відео [2, 3, 4]. Ці документи зберігають повний протокол обчислень і їх результатів та можуть бути легко опубліковані у веб. Зараз IPython Notebook є частиною більш масштабного проекту Jupyter.

Відома складність розробки та супроводження інформаційних систем, які використовують велику кількість даних, що відрізняються типом і структурою. До таких систем можна віднести системи автоматизованого проектування, управління та підтримки прийняття

рішень. Одним із способів вирішення цієї проблеми є побудова програми з незалежних програмних компонентів, наприклад класів, відповідно до об'єктно-орієнтованої парадигми програмування [1, 5]. В цих класах можуть бути інтегровані дані, алгоритми їх опрацювання, документація та інтерфейси користувача. Python володіє ефективними засобами розробки таких компонентів, але на даний час не описані особливості їх реалізації для застосування в IPython Notebook.

Мета даної роботи - на простому прикладі показати принципи створення бібліотеки таких компонентів та її використання в інтерактивному обчислювальному середовищі IPython Notebook.

Розробимо модуль `normTPdatabase.py`, який містить компоненти для розрахунку норм часу на технологічну операцію механічної обробки деталей. Компоненти реалізуємо за допомогою класів мови Python. Нижче показано код класу `DB`, призначеного для розрахунку основного часу для точіння, розточування або свердління. Клас містить рядок документації з розміткою Markdown та вставкою формули LaTeX. Метод класу `doc`, який належить класу, а не об'єкту, призначений для візуалізації документації Markdown в Notebook. Спеціальний метод `__init__` (конструктор) викликається під час створення об'єкта класу та ініціалізує атрибути `l`, `n`, `s`. Метод `time` розраховує та повертає значення основного часу.

```
from IPython.display import display, Markdown, HTML
class DB:
    """Основний час для точіння, розточування, свердління
     $T_o = \frac{l}{n \cdot s}$ 
    de
    *l* - довжина робочого ходу
    *n* - частота обертів
    *s* - подача"""
    @classmethod
    def doc(cls):
        return Markdown(cls.__doc__.decode("utf-8"))
    def __init__(self, l=None, n=None, s=None):
        self.l=l
        self.n=n
        self.s=s
```

```
def time(self):
    return self.l/(self.n*self.s)
```

Цей клас додається в db - список, який міститиме усі класи модуля.

```
db=[]
db.append(DB)
```

Наступний клас призначений для визначення допоміжного часу, який затрачається на установку заготовки. Цей клас також містить рядок документації, метод класу doc, конструктор та метод time. Крім цього, клас має структуру даних data з даними, необхідними для визначення допоміжного часу.

```
class DB:
    """Час на установку і зняття заготовки в самоцентруючому патроні
    m - маса
    k - mun (список munів: get_doc(db[i].data)):
    """
    data=[
        ("Самоцентруючий патрон, кріплення ключем",{0.5:0.15, 1:0.17, 3:0.23,
        5:0.12}),
        ("Самоцентруючий патрон, пневмозатиск",{0.5:0.07, 1:0.08, 3:0.1,
        5:0.27})]
    @classmethod
    def doc(cls):
        n=cls.__doc__+"\n"
        d=""
        for r in cls.data:
            c=""<td>%s</td>"%r[0]
            for k in sorted(r[1].keys()):
                c+=""<td>m=%s t=%s</td>"%%(k,r[1][k])
            d+=""<tr>%s</tr>"%c
        d=""<table>%s</table>"%d
        return
    display(Markdown(n.decode("utf-8")),HTML(d.decode("utf-8")))
    def __init__(self, k=None, m=None):
```

```

self.k=k
self.m=m
def time(self):
    row=self.data[self.k][1]
    keys=sorted(row.keys())
    keys2=[x for x in keys if x<=self.m]
    if keys2: key=keys2[-1]
    else: key=keys[0]
    return row[key]

```

Цей клас теж додається в список db.

db.append(DB)

Можна створити в цьому класі методи для роботи з графічним інтерфейсом користувача (GUI) - `gui` та `gui_ok`. Це полегшить введення даних користувачем. Нижче наведено код цих методів. Метод `gui` створює вікно для вводу користувачем необхідних даних. Метод `gui_ok` викликається після натиску на кнопку `Ok`, присвоює атрибутам `k` і `m` введені користувачем значення, знищує вікно та виходить з інтерпретатора `Tcl`. Для побудови GUI використовується стандартний модуль `Tkinter`.

```

def gui(self):
    root=Tkinter.Tk()
    root.attributes("-topmost", True)
    self.list=Tkinter.Listbox(root)
    listData=[k[0] for k in self.data]
    for x in listData:
        self.list.insert(Tkinter.END,x)
    self.list.pack()
    self.entry=Tkinter.Entry(root)
    self.entry.pack()
    btn=Tkinter.Button(root,text="Ok",command=self.gui_ok)
    btn.pack()
    self.root=root
    root.mainloop()
def gui_ok(self):

```

```
self.k=int(self.list.cursorselection()[0])
self.m=float(self.entry.get())
self.root.destroy()
self.root.quit()
```

Вкінці модуля автоматично генерується рядок документації модуля `__doc__`.

```
__doc__="Введіть db[i] та db[i].__doc__ для допомоги,  
де i:\n"  
docList=[str(i)+" - "+x.__doc__.split("\n")[0] for i,x in enumerate(db)]  
__doc__+="\n".join(docList)
```

Тепер створимо документ Notebook для розрахунку норми часу на задану операцію механічної обробки. Щоб розпочати роботу з IPython Notebook в командному рядку введіть: `ipython notebook`. Відкриється браузер з домашньою сторінкою, на якій можна створити новий документ (меню `New/Python 2`) або завантажити існуючий. Введемо після запрошення (In []:) потрібний код мовою Python та натиснемо `Ctrl+Enter`, щоб виконати його. Для додання нової комірки користуємось меню `Insert`. Імпортуємо з модуля `normTPdatabase` рядок документації модуля `__doc__` і список `db` з класами для визначення норм часу (рис. 1). Виводимо рядок документації модуля. В модулі `normTPdatabase` він може генеруватись автоматично за рядками документації кожного класу в списку `db`. Таким чином ми дізнались індекс класу основного часу в списку `db` - 0. Тепер введемо документацію для цього класу. Якщо документація виводиться в Notebook, то вона може включати вставки мовами розмітки `LaTeX`, `Markdown`, `HTML`. Візуалізація формул `LaTeX` виконується за допомогою скриптів `MathJax`.

Після перегляду документації ми можемо створити об'єкт класу `db[0]`, передати дані конструктору та відразу ж викликати метод `time` (рис. 2), який розраховує основний час. В комірці `Out[4]` бачимо результат. Тепер введемо документацію для класу `db[1]`, який призначений для розрахунку допоміжного часу.

```
In [1]: from normTPdatabase import __doc__, db
```

```
In [2]: print __doc__
```

Введіть db[i] та db[i].__doc__ для допомоги, де i:
 0 - Основний час для точіння, розточування, свердління
 1 - Час на установку і зняття заготовки в самоцентруючому патроні

```
In [3]: db[0].doc()
```

Out[3]: Основний час для точіння, розточування, свердління

$$T_o = \frac{l}{ns}$$

де

l - довжина робочого ходу

n - частота обертів

s - подача

Рис. 1. Документ IPython Notebook

В даному випадку документація містить вставки мовою розмітки HTML, що дозволяє нарисувати таблицю. Вміст таблиці генерується автоматично методом doc за атрибутом класу data, який містить дані для вибору допоміжного часу в залежності від маси заготовки і типу закріплення. Переглянути ці дані можна командою print db[1].data або print repr(db[1].data).decode('string_escape'). Тепер створимо об'єкт t класу db[1] та виконаємо метод gui, який будує вікно для введення типу закріплення k і маси заготовки m. Після натискання кнопки Ok на цьому вікні, об'єкт отримає необхідні значення k і m. Після виклику методу time для об'єкта t, в комірці Out[6] бачимо результат розрахунку допоміжного часу (рис. 3).

Покажемо можливості IPython Notebook для роботи з графічною бібліотекою для побудови графіків Matplotlib [6]. Побудуємо залежності сумарного оперативного часу від довжини L циліндричної деталі, яка обробляється на токарному верстаті. Будуть побудовані дві такі залежності, для двох технологій, які відрізняються швидкістю подачі ($s=0,1$ і $s=0,2$) та способом установки заготовки ($k=1$ і $k=0$). Спочатку опишемо функцію mass, для розрахунку маси циліндричної деталі за її діаметром d, довжиною l і густиною ρ .

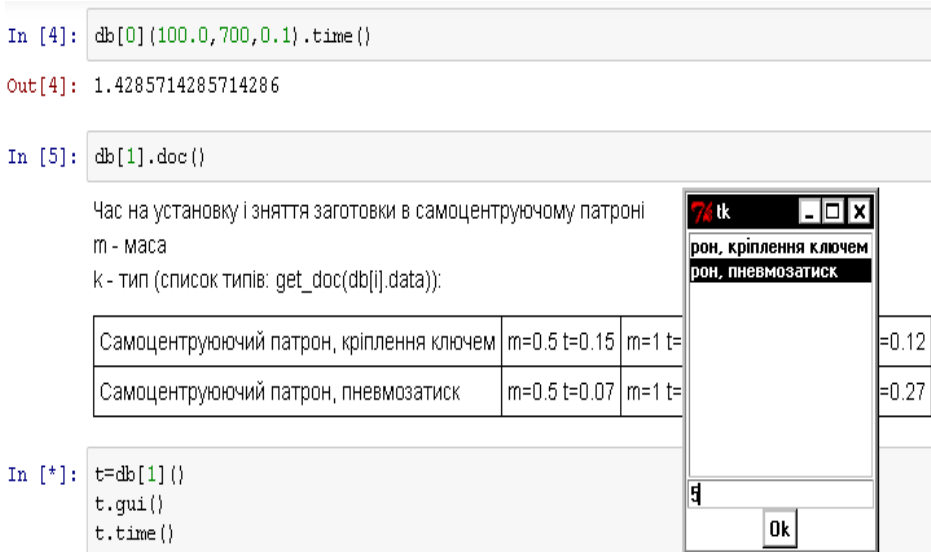


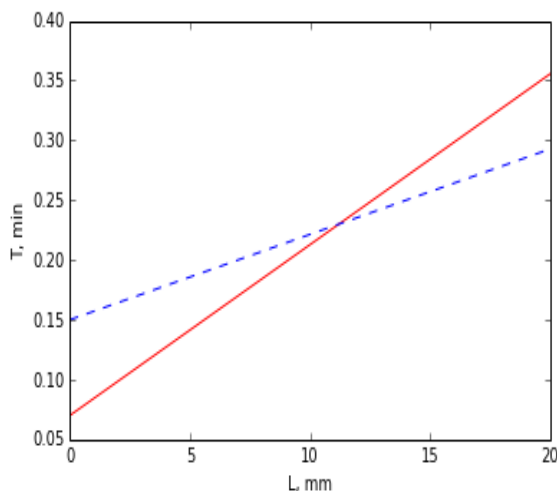
Рис. 2. Документ IPython Notebook (продовження)

Для побудови графіків Matplotlib у вікні Notebook, необхідно ввести команду `%matplotlib inline`. Списки T1 та T2 містять два значення сумарного часу для двох технологій. Перше відповідає довжині деталі 0 мм, друге - 20 мм. Графік (рис. 3) дозволяє визначити інтервали довжин L деталі, на яких ефективно задана технологія. Бачимо, що на інтервалі 0...11 мм ефективнішою буде перша технологія, так як вона виконується за менший оперативний час.

Out[6]: 0.27

```
In [7]: def mass(d,l,ro):
        return ro*3.14159*1*(d/2.0)**2
```

```
In [8]: %matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
L=[0.0,20.0]
T1=[0,0]
T1[0]=db[0](L[0],700,0.1).time()+db[1](1,mass(100.0,L[0],0.000001)).time()
T1[1]=db[0](L[1],700,0.1).time()+db[1](1,mass(100.0,L[1],0.000001)).time()
T2=[0,0]
T2[0]=db[0](L[0],700,0.2).time()+db[1](0,mass(100.0,L[0],0.000001)).time()
T2[1]=db[0](L[1],700,0.2).time()+db[1](0,mass(100.0,L[1],0.000001)).time()
plt.plot(L,T1,'r',L,T2,'b')
plt.xlabel('L, mm')
plt.ylabel('T, min')
plt.show()
```



In []:

Рис. 3. Документ IPython Notebook (продовження)

Висновки. Показані компоненти є незалежними та володіють високим рівнем інтеграції даних, алгоритмів і документації. Документація генерується автоматично та може містити розширене форматування. Компоненти можуть бути використані в програмах з інтерфейсом командного рядка та в програмах з GUI. Це, разом з іншими потужними можливостями Python, дозволить спростити процес розробки та супроводження програм. Зокрема, автор планує застосування цих принципів в системі підтримки прийняття рішень для проектування штангових свердловинних насосних установок [7].

Показаны принципы разработки библиотеки программных компонентов на языке Python и ее использования в интерактивной вычислительной среде IPython Notebook. Компоненты реализуются Python-классами, независимы и обладают высоким уровнем интеграции данных, алгоритмов, пользовательских интерфейсов и автодокументации с расширенным форматированием.

Ключевые слова: программный компонент, Python, IPython Notebook, интерактивная вычислительная среда, объектно-ориентированное программирование, класс

The purpose of this work - by a simple example to show the principles of developing the library of software components in Python and use it in an interactive computational environment IPython Notebook. Components are implemented by Python classes, are independent and have a high level of data integration, algorithms, user interfaces and documentation. Documentation is automatically generated and may contain advanced formatting. The components can be used in programs with a command line interface and with a graphical user interface. Together with the powerful Python features this simplifies the process of software development and software maintenance. The proposed principles can be used to develop and maintain information systems that use large amounts of data with different type and structure, such as computer-aided design, automated control systems, decision support systems.

Keywords: software component, Python, IPython Notebook, interactive computational environment, object-oriented programming, class

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бизли Д. Python. Подробный справочник / Дэвид Бизли. - СПб.: Символ-Плюс, 2010. - 864 с.
2. Pérez, Fernando. IPython: A System for Interactive Scientific Computing / Fernando Pérez, Brian E. Granger // Computing in Science and Engineering, vol. 9, no. 3, May/June 2007. - pp. 21-29.
3. Martins, L. Felipe. IPython Notebook Essentials / L. Felipe Martins. - Birmingham: Packt Publishing, 2014. - 190 p. - ISBN 978-1-78398-834-1.
4. Rossant, Cyrille. IPython Interactive Computing and Visualization Cookbook / Cyrille Rossant. - Birmingham: Packt Publishing, 2014. - 512 p. - ISBN 978-1-78328-481-8.
5. Phillips, Dusty. Python 3 Object-oriented Programming, 2nd Edition / Dusty Phillips. - Birmingham: Packt Publishing, 2015. - 460 p. - ISBN 978-1-78439-878-1.
6. Hunter, J.D. Matplotlib: A 2D Graphics Environment / J.D. Hunter // Computing in Science & Engineering, vol.9, no.3, May-June 2007. - pp. 90-95.
7. Копей, В.Б. Застосування мови програмування Python для побудови баз знань з проблем надійності і довговічності штангових свердловинних насосних установок / В.Б. Копей, І.І. Палійчук // Нафтогазова енергетика. - №2(15). - 2011. - С.12-18.