Державний вищий навчальний заклад
«Київський національний економічний університет імені Вадима Гетьмана»
Фаховий коледж інформаційних систем і технологій

«Модульність в Python»

Викладач дисципліни «Алгоритмізація та програмування»
Стефанцев Сергій Сергійович

КИЇВ - 2021

Мета заняття

- Ознайомитися із модульністю в Python.
- Навчитися використовувати математичні функції, імпортувати з модулів, імпортувати окремі функції з модуля, створювати власні модулі. Використовувати каталоги пошуку модулів, пакети.

План заняття

1. Модульність в Python.
2. Модуль math. Математичні функції.
3. Імпорт з модулів та його види.
4. Імпорт окремої функції з модуля.
5. Створення власних модулів.
6. Каталоги пошуку модулів.
7. Пакети.

Модульність в Python

Якщо код програми є складним, розбиття її на окремі функції допомагає спростити його для візуального сприйняття. Якщо цього недостатньо, є сенс винести частину функцій та пов'язаних з ними оголошень за межі основного файлу програми. Такі додаткові файли з кодом, що використовується в програмі, називаються модулями. Найчастіше вони містять оголошення функцій та констант, які далі можуть бути підключені (імпортовані) в головну програму і вільно в ній використовуватися. Об'єкти з модуля можуть бути імпортовані в інші модулі. Файл утворюється шляхом додавання до імені модуля розширення .py. При імпорті модуля інтерпретатор шукає файл спочатку в поточному каталозі, потім в каталогах, зазначених у змінній оточення PYTHONPATH, потім в залежних від платформи шляхах за замовчуванням, а також в спеціальних файлах з розширенням '.pth', які лежать в стандартних каталогах.

Модульність в Python

Можна внести зміни в PYTHONPATH і в ‘.pth’, додавши туди свій шлях. Каталоги, в яких здійснюється пошук, можна подивитися в змінній sys.path. Великі програми, як правило, складаються з стартового файлу – файлу верхнього рівня, і набору файлів-модулів. Головний файл займається контролем програми. У той же час модуль – це не тільки фізичний файл. Модуль являє собою колекцію компонентів. У цьому сенсі модуль – це простір імен, – namespace, і всі імена всередині модуля ще називаються атрибутами – такими, наприклад, як функції і змінні. Є велика кількість вбудованих модулів, що дозволяють виконувати складні математичні операції (math), працювати з датами (datatime)/часом (time), випадковими числами (random), операційною та файловою системами (os) та ін.

Модуль math. Математичні функції

Модуль math надає додаткові функції для роботи з числами, а також стандартні константи. Перш ніж використовувати модуль, необхідно підключити його за допомогою інструкції:

import math

Модуль *math* надає наступні стандартні константи:

pi – повертає число *n*.

e – повертає значення константи *e*.

Модуль math. Математичні функції


```
import math

print(math.pi)
print(math.e)
```

```
3.141592653589793
2.718281828459045
```

Модуль math. Математичні функції

Основні функції для роботи з числами:

sin (), cos (), tan () – стандартні тригонометричні функції (синус, косинус, тангенс). Значення вказується в радіанах;

asin (), acos (), atan () – зворотні тригонометричні функції (арксинус, арккосинус, арктангенс). Значення повертається в радіанах;

degrees() – перетворює радіани в градуси:

```
import math
print(math.degrees(math.pi))
```

```
180.0
```

Модуль math. Математичні функції

radians() – перетворює градуси в радіани:

```
import math
print(math.radians(180.0))
```

```
3.141592653589793
```

exp() – експонента;
log() – логарифм;
sqrt() – квадратний корінь:

```
import math
print(math.sqrt(100), math.sqrt(25))
```

```
10.0 5.0
```

Модуль math. Математичні функції

ceil() – значення, округлене до найближчого більшого цілого:

```
import math
print(math.ceil(5.49), math.ceil(5.50), math.ceil(5.51))
```

```
6 6 6
```

floor() – значення, округлене до найближчого меншого цілого:

```
import math
print(math.floor(5.49), math.floor(5.50), math.floor(5.51))
```

```
5 5 5
```

Модуль math. Математичні функції

pow(Число, Степень) – підносить Число до Степені:

```
import math
print(math.pow(10, 2), 10 ** 2, math.pow(3, 3), 3 ** 3)
```

```
100.0 100 27.0 27
```

fabs() – абсолютне значення:

```
import math
print(math.fabs(10), math.fabs(-10), math.fabs(-12.5))
```

```
10.0 10.0 12.5
```

Модуль math. Математичні функції

fmod() – остача від ділення:

```
import math
print(math.fmod(10, 5), 10 % 5, math.fmod(10, 3), 10 % 3)
```

```
0.0 0 1.0 1
```

factorial() – факторіал числа:

```
import math
print(math.factorial(5), math.factorial(6))
```

```
120 720
```

Модуль random. Випадкові числа

Більшість програм роблять одне і те ж при кожному виконанні, тому говорять, що такі програми визначені. Визначеність хороша річ до тих пір, поки ми вважаємо, що одні й ті ж обчислення повинні давати один і той же результат. Проте, в деяких програмах від комп'ютера потрібно непередбачуваність. Типовим прикладом є ігри, але є маса інших застосувань: зокрема, моделювання фізичних процесів або статистичні експерименти. Змусити програму бути дійсно непередбачуваною завдання не таке просте, але є способи змусити її здаватися непередбачуваною. Одним з таких способів є генерування випадкових чисел і використання їх у програмі. У Python є вбудований модуль, який дозволяє генерувати псевдовипадкові числа. З математичної точки зору, вони не істинно випадкові.

Модуль random. Випадкові числа

Модуль random дозволяє генерувати випадкові числа. Перш ніж використовувати модуль, необхідно підключити його за допомогою інструкції:

import random

Основні функції:

random() – повертає псевдовипадкове дійсне число від 0.0 до 1.0:

```
import random
print(random.random())
print(random.random())
print(random.random())
```

```
0.4597989268795628
0.21393781572951398
0.30104337991190655
```

Модуль random. Випадкові числа

Числа, що видаються функцією *random()*, розподілені рівномірно – це означає, що всі значення рівноймовірні.

uniform(start, end) – повертає псевдовипадкове дійсне число в діапазоні від start до end:

```
import random
print(random.uniform(0, 10))
print(random.uniform(0, 10))
```

```
3.8573686038924127
1.0758724939305075
```

randint(start, end) – повертає псевдовипадкове ціле число в діапазоні від start до end:

```
import random
print(random.randint(0, 10))
print(random.randint(0, 10))
```

```
3
8
```

Модуль random. Випадкові числа

randrange(start, end, step) – повертає випадковий елемент з числової послідовності. Параметри аналогічні параметрам функції *range()*. Саме зі списку, що повертається функцією *range()*, і вибирається випадковий елемент:

```
import random
print(random.randrange(10))
print(random.randrange(0, 10))
print(random.randrange(0, 10, 2))
```

```
0
6
8
```

Модуль random. Випадкові числа

choice(Послідовність) – повертає випадковий елемент з будь-якої послідовності (рядку, списку, кортежу):

```
import random
print(random.choice("string")) # Випадковий символ з рядку
print(random.choice(["s", "t", "r"])) # Випадковий елемент зі списку
print(random.choice(("s", "t", "r"))) # Випадковий елемент з кортежу
```

```
i
s
r
```

Модуль random. Випадкові числа

shuffle(Список, Число від 0.0 до 1.0) – перемішує елементи списку випадковим чином. Функція перемішує сам список і нічого не повертає. Якщо другий параметр не вказано, то використовується значення, яке повернене функцією random().

```
import random
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
random.shuffle(lst)
print(lst)
```

```
[2, 7, 3, 10, 6, 9, 5, 4, 1, 8]
```

Модуль random. Випадкові числа

sample(Послідовність, Кількість елементів) – повертає список із зазначеної кількості елементів. У цей список потраплять елементи з послідовності, вибрані випадковим чином. Як послідовність – можна вказати будь-який об'єкт, що підтримує ітерації.

```
import random
print(random.sample("string", 2))
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(random.sample(lst, 2))
print(lst) # Сам список не змінюється
print(random.sample((1, 2, 3, 4, 5, 6, 7), 3) )
```

```
['t', 's']
[4, 5]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[7, 1, 4]
```

Імпорт з модулів та його види

Для того щоб отримати доступ до функцій або змінних/констант з модуля та використати їх в основній програмі – його необхідно підключити до програми. Це можна зробити за допомогою інструкції import МОДУЛЬ, де модуль це ім’я іншого файлу Python без розширення .py.

import math

Дана команда імпортує модуль math. Тепер необхідно викликати з нього одну з функцій. Для того, щоб звернутися до змінної або функції з імпортованого модуля необхідно вказати його ім'я, поставити крапку і вказати необхідне ім’я **МОДУЛЬ.ФУНКЦІЯ/КОНСТАНТА**.

```
import math
print (math.e)
```

```
2.718281828459045
```

Імпорт з модулів та його види

Дізнатися, які функції і константи визначені в модулі можна за допомогою функції ***dir()*** :

```
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

В результаті виконання цієї команди інтерпретатор вивів всі імена, визначені в цьому модулі. У їх числі є і змінна ***___doc___,*** що дозволяє вивести опис модуля.

```
import math
print (math.__doc__)
```

```
This module provides access to the mathematical functions
defined by the C standard.
```

Імпорт з модулів та його види

___doc___ є внутрішнім ім'ям рядка документації як змінної всередині функції.

```
import math
print (math.e.__doc__)
```

```
Convert a string or number to a floating point number, if possible.
```

Крім того, дізнатися про функції, які містить модуль, можна через функцію ***help()***:

```
import math
print(help(math))
```

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

    asinh(x, /)
```

Імпорт окремої функції з модуля

В Python можна імпортувати окрему функцію з модуля за допомогою наступної конструкції:

from МОДУЛЬ import ФУНКЦІЯ/КОНСТАНТА

```
from math import sqrt
print(sqrt(9))
```

```
3.0
```

Таким чином, Python не створюватиме змінну *math*, а завантажить в пам'ять тільки функцію *sqrt()*. Тепер виклик функції можна робити, не звертаючись до імені модуля *math*.

Імпорт окремої функції з модуля

Через кому можна перерахувати декілька функцій або змінних які необхідні.

```
from math import sqrt, factorial
print(sqrt(9))
print(factorial(9))
```

```
3.0
362880
```

Або вказати * і тоді можна звертатися до будь-яких функцій.

```
from math import *
print(sin(pi/2))
```

```
1.0
```

Імпорт окремої функції з модуля

В якості параметра тригонометричні функції приймають значення кута в радіанах.

```
from math import *
print(help(sin))
```

```
Help on built-in function sin in module math:

sin(x, /)
    Return the sine of x (measured in radians).

None
```

Створення власних модулів

Щоб створити власний модуль необхідно зберегти файл з власним ім'ям **ІМ'Я.ру** (для модулів обов'язково вказується розширення *.ру*), що містить якийсь код (вміст модуля). Наприклад створимо модуль назвавши його *my_math*:

```
def my_func ():
    print('test')
```

```
import my_math                  # Імпорт модуля my_math

my_math. my_func()              # Виклик функції my_func
```

Результатом запуску даного коду буде:

```
test
```

Створення власних модулів

Якщо необхідно імпортувати одноіменні функції з кількох модулів, для них можна задати псевдоніми. Тоді функція буде теж скопійована в поточний простір імен, але під іншою назвою за допомогою наступної конструкції:

from МОДУЛЬ import ФУНКЦІЯ/КОНСТАНТА as НОВЕ_ІМ'Я

Створення власних модулів

Імпортування модуля виконує команди що містяться в ньому. Однак, повторне імпортування не приводить до виконання модуля, тобто він повторно не імпортується. Пояснюється це тим, що імпортування модулів в пам'ять – ресурсномісткий процес, тому зайвий раз Python його не виконує. Якщо було внесено зміни до модуля – необхідно його повторно імпортувати, примусово вказати Python, що модуль вимагає повторного завантаження. Після виклику функції ***reload()*** із зазначенням в якості аргументу імені модуля, оновлений модуль завантажиться повторно.

```
import imp

imp.reload(my_math)
```

Результатом запуску даного коду буде:

```
test
<module 'mtest' from 'C:\\Python35-32\\mtest.py'>
```

Каталоги пошуку модулів

Для імпорту Python шукає файли, що зберігається в стандартному модулі sys, як змінну path. Можна отримати доступ до цього списку і змінити його (відрізняється для різних операційних систем).

```
import sys

for place in sys.path:    # path містить список шляхів пошуку модулів
    print(place)
```

Результатом запуску даного коду буде:

```
D:\Users\syad\AppData\Local\Programs\Python\Python35\Lib\idlelib
D:\Users\syad\AppData\Local\Programs\Python\Python35\python35.zip
D:\Users\syad\AppData\Local\Programs\Python\Python35\DLLs
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib
D:\Users\syad\AppData\Local\Programs\Python\Python35
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib\site-packages
```

Каталоги пошуку модулів

Список sys.path містить шляхи пошуку, одержувані з наступних джерел:

– шлях до поточного каталогу з виконуваним файлом;

– значення змінної оточення PYTHONPATH. Для додавання змінної в меню Пуск необхідно обрати пункт Панель керування (або Налаштування | Панель управління). Обрати пункт Система. Перейти на вкладку Додатково і натиснути кнопку Змінні середовища. У розділі Змінні середовища користувача натиснути кнопку Створити. В поле Ім’я змінної ввести "PYTHONPATH", а в полі Значення змінної задати шлях до папок, модулів через крапку з комою.

Каталоги пошуку модулів

Після цих змін перезавантажувати комп'ютер не потрібно, достатньо заново запустити програму;

– шляхи пошуку стандартних модулів;

– вміст файлів з розширенням pth, розташованих в каталогах пошуку стандартних модулів, наприклад, в каталозі D:\Users\syad\AppData\Local\Programs\Python\Python35\lib\sitepackages. Назва файлу може бути довільною, головне, щоб розширення файлу було pth. Кожен шлях (абсолютний або відносний) повинен бути розташований на окремому рядку.

Каталоги повинні існувати, в іншому випадку вони не будуть додані в список sys.path.

Каталоги пошуку модулів

При пошуку модуля список *sys.path* проглядається зліва направо. Пошук припиняється після першого знайденого модуля. Таким чином, якщо в каталогах D:\Users\\folder1 і D:\Users\folder2 існують однойменні модулі, то буде використовуватися модуль з папки D:\Users\folder1, оскільки він розташований першим у списку шляхів пошуку. Список *sys.path* можна змінювати з програми за допомогою спискових методів. Наприклад, додати каталог в кінець списку можна за допомогою методу *append()*.

Каталоги пошуку модулів


```
import sys

sys.path.append(r" D:\Users\folder1")          # Додаємо в кінець
списку
for place in sys.path:
    print(place)
```

Результатом запуску даного коду буде:

```
D:\Users\syad\AppData\Local\Programs\Python\Python35\Lib\idl
elib
D:\Users\syad\AppData\Local\Programs\Python\Python35\python
35.zip
D:\Users\syad\AppData\Local\Programs\Python\Python35\DLLs
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib
D:\Users\syad\AppData\Local\Programs\Python\Python35
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib\site
-packages
D:\Users\folder1
```

Каталоги пошуку модулів

Додати каталог в початок списку – за допомогою методу ***insert().***

```
import sys

sys.path.insert(0, r" D:\Users\folder2")        # Додаємо на
початку списку
for place in sys.path:
    print(place)
```

Результатом запуску даного коду буде:

```
D:\Users\folder2

D:\Users\syad\AppData\Local\Programs\Python\Python35\Lib\idl
elib
D:\Users\syad\AppData\Local\Programs\Python\Python35\python
35.zip
D:\Users\syad\AppData\Local\Programs\Python\Python35\DLLs
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib
D:\Users\syad\AppData\Local\Programs\Python\Python35
D:\Users\syad\AppData\Local\Programs\Python\Python35\lib\site
-packages
D:\Users\folder1
```

Каталоги пошуку модулів

Символ *r* перед лапками дозволяє не інтерпретувати спеціальні послідовності. Якщо використовуються звичайні рядки, то необхідно подвоїти кожен слеш в шляху:

```
sys.path.append ("D:\\Users\\folder1\\folder2\\folder3")
```

Пакети

Коли модулів стає забагато, виникає необхідність групувати їх далі. Для цього файли модулів розкладаються по папках. Відомо, що інтерпретатор шукає модулі в поточній папці та у спеціально призначеному для цього місці, отже необхідно якимось чином показати йому, що папка поряд з вашою програмою – не просто папка з файлами, а містить модулі для підключення. Для цього в папці повинен знаходитися файл *__init__.py* – він може бути порожнім, але сама його наявність сигналізує інтерпретатору, що папка із ним є пакетом модулів і може використовуватися в програмі.

Пакети

Пакетом називається каталог з модулями, в якому розташований файл ініціалізації ***__init__.py***.

```
import my_package.my_math
# Модуль my_math шукатиметься в пакеті my_package

print(my_package.my_math.exp(1))
```

Отримаємо:

```
2.718281828459045
```

Всі розглянуті види імпорту поширюється також на пакети. Лише в іменах додається додатковий елемент через крапку – назва пакету. Пакети можуть вкладатися в інші пакети, аналогічно додаючи нові простори імен. Як і модулі, пакети можуть містити код, який буде виконано під час ініціалізації пакету, – він записується в самому файлі *__init__.py*

Використана література

1. Язык программирования Python / Г. Россум, Ф. Л. Дж. Дрейк, Д. С. Откидач та ін. 2001. – 454 с.
2. Любанович Б. Python. Простой Python. Современный стиль программирования / Б. Любанович. – СПб. : Питер, 2016. – 480 с.
3. Федоров Д. Ю. Основы программирования на примере языка Python : учеб.пособие / Д. Ю. Федоров. – СПб. : Питер, 2016. – 176 с.
4. Прохоренок Н. А. Python 3 и PyQt. Разработка приложений / Н. А. Прохоренок. – СПб. : БХВ-Петербург, 2012. – 704 с.

ДЯКУЮ ЗА УВАГУ!