Лекція 18
Модулі і пакети

Контрольна робота №7 (визначення варіанту)

ІО	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
61	15	16	1	2	3	4	5	6	7	8	9	10	11	12	13
62	14	15	16	1	2	3	4	5	6	7	8	9	10	11	12
63	13	16	15	14	1	2	3	4	5	6	7	8	9	10	11
64	12	7	14	15	16	10	2	3	4	5	6	13	8	9	1
65	11	12	13	14	15	16	1	2	3	4	5	6	7	8	9

Червоний колір – номер у списку групи

Чорний та синій колір – номер варіанту

ІО	16	17	18	19	20	21	22	23	24	25	26	27	28	29
61	10	11	12	13	14	15	16	1	2	3	4	5	6	7
62	8	9	10	11	12	13	14	15	16	1	2	3	4	5
63	6	7	8	9	10	11	12	13	14	15	16	1	2	3
64	4	5	6	7	8	9	10	11	12	13	14	15	16	1
65	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Контрольна робота 7. Функції користувача

Написати програму, що містить виклик функції, для отримання зазначеного результату:

1	def func(*t, a, b=10, **d):
	print(t, a, b, d)
	Результат: (35, 10) 1 10 {'c': 3}
		2	def func(x, y):
			for i in range(1, x+1):
			yield i ** y
			Результат: 1 8 27 64 125

3	def s(a=2, b=3, c=4): return a + b + c Результат: rezult 20	4	def sum(x, y): return x + y Результат: рядок
5	def rizn (x, y) : return x - y Результат: 30 – 10 = 20	6	def rizn (x, y, z=2): return x - y + z Результат: a = 17
7	def s(a=3, b=2, c=4): return a + b + c Результат: rezult 9	8	def s(a=2, b=3, c=4): return a + b + c Результат: rezult 25
9	f = lambda x, y = 2: x + y Результат: 7	10	f = lambda x, y = 2: x + y Результат: 11

11	def suma(a, b, c):
	    return a + b + c
	d1 = {"a": 1, "b": 2, "c": 3}
	Результат: rez 6
		12	def suma (a, b, c):
			    return a + b + c
			t = (3,4,5)
			Результат: 12

13	def suma(x, y=5, *t):
	    res = x + y
	    for i in t:
	        res += i
	    return res
	Результат: rez 100
		14	def suma(*t):
			    res = 0
			    for i in t:
			        res += i
			    return res
			Результат: 100

15	def suma(a, b, c):
	    return a + b + c
	t, d2 = (1, 2), {"c": 3}
	Результат: rez 6
		16	def suma (a, b, c):
			    return a + b + c
			t1,t2,t3 = (1, 2),(3, 4),(5, 6)
			Результат: (1, 2, 3, 4, 5, 6)

Означення модуля

Модулем у мові Python називають будь-який файл з програмним кодом.

Кожний модуль може імпортувати інший модуль.

Після імпортування одержуємо доступ:

1. до атрибутів,
2. до змінних,
3. до функцій,
4. до класів,

які оголошених усередині імпортованого модуля.

Імпортувати можемо модулі,які написані іншими мовами

Імпортований модуль може містити програму не тільки мовою Python – так, можна імпортувати скомпільований модуль, написаний мовою C.

Модуль "__main__"

Усі програми, які ми запускали раніше, були розташовані в модулі з назвою "__main__".

Одержати ім'я модуля дозволяє визначений атрибут __name__. Виведемо назву модуля:

```
print(_name_)
```

Результат роботи: __main__

Перевірка, чи є модуль головною програмою або імпортованим модулем

Приклад 1. Перевірка способу запуску модуля

```
if __name__=="__main__":
    print("Це головна програма")
 else:
    print( "Імпортований модуль")
```

```
Результат виконання: Це головна програма
```

Приклад у файлах: example1.py та testex1.py

Інструкція `import`

Імпортувати модуль дозволяє інструкція `import`.
Інструкція `import` має наступний формат:

import <Назва модуля1> [<as Псевдонім1>]
[,... ,<Назва модуляN> [<as ПсевдонімN>]]

1.Після ключового слова `import` вказують назва модуля.

```
import os
```

2.Назва модуля не повинна містити розширення й шляху до файлу.

3.Назва модуля повинна повністю відповідати правилам іменувань змінних.

4.Крім того, слід уникати збігу імен модулів з ключовими словами, вбудованими ідентифікаторами й назвами модулів, що входять у стандартну бібліотеку.

```
import math

import time

import mytest
```

Імпортування кількох модулів однією інструкцією `import`

За один раз можна імпортувати відразу кілька модулів, записавши їх через кому. Для прикладу підключимо модулі `time` і `math`

Приклад 2.

```
import time, math # Імпортуємо кілька модулів відразу

print (time.strftime ( "%d. % m. % y"))
# Поточна дата
print(math.pi) # Число pi
Результат виконання:
25.11.16
3.141592653589793
```

Правила доступу до атрибутів у імпортованому модулі

Після імпортування модуля його назва стає ідентифікатором.

Через ідентифікатор можна одержати доступ до атрибутів, визначених усередині модуля.

Доступ до атрибутів модуля здійснюється за допомогою точкової нотації.

Наприклад, звернутися до констант `pi` та `e`, розташованих усередині модуля `math`, можна так:

```
math.pi
math.e
```

Функція `getattr()`

Функція **`getattr()`** дозволяє одержати значення атрибута модуля по його назві, заданій у вигляді рядка.

За допомогою цієї функції можна сформувати назву атрибута динамічно під час виконання програми.

Формат функції:

getattr(<Объект модуля>, <Атрибут>[, <Значення за замовчуванням>])

Якщо зазначений атрибут не знайдений, виконується виключення `Attributeerror`. Щоб уникнути виводу повідомлення про помилку, можна в третьому параметрі вказати значення, яке буде повертатися, якщо атрибут не існує.

Приклад використання функції getattr

Приклад 2.

```
import math

# Число pi
print (getattr (math, "pi" ))

# число 50, тому що x не існує
print (getattr (math, "x", 50))

Результат виконання:
3.141592653589793
50
```

Приклад у файлах: example2.py та testex2.py

Функція hasattr()

Функція **hasattr()** дозволяє перевірити існування атрибута модуля. Формат функції:

hasattr (<Об'єкт>, <"Назва атрибута">).

Якщо атрибут існує, функція повертає значення True.

Приклад 3.

```
import math
def chattr_math(attr):
    if hasattr(math, attr):
        return "Атрибут існує"
    else:
        return "Атрибут не існує"
print (chattr_math("pi")) # Атрибут існує
print (hasattr_math ("x")) # Атрибут не існує
Результат виконання:
Атрибут існує
Атрибут не існує
```

Приклад у файлах:
example3.py та testex3.py

Псевдонім модуля

Псевдонім задають після ключового слова **as**.

Приклад 4.

```
import math as m # Створення псевдоніма
print (m.pi) # Число pi
Результат виконання:
3.141592653589793
```

1.Тепер доступ до атрибутів модуля `math` може здійснюватися тільки за допомогою ідентифікатора `m`.

2.Ідентифікатор `math` у цьому випадку використовувати вже не можна.

3.Увесь уміст імпортованого модуля доступний тільки через ідентифікатор, зазначений в інструкції `import`.

Простори імен

Будь-яка глобальна змінна насправді є глобальною змінною модуля.

Тому модулі використовують як простори імен.

Приклад формування простору імен

1.Створимо модуль з назвою `tests.py`, у якому визначимо змінну `x`.

```
x = 50
```

2.В основній програмі також визначимо змінну `x`, але з іншим значенням.

3.Потім підключимо файл `tests.py` і виведемо значення змінних.

Приклад 5.

```
import tests  # Підключаємо файл tests.py

x = 22
print(tests.x)  # Значення змінно x усередині мод

print(x) #   Значення змінної x в основній програмі

Результат виконання: 50 22
```

Ніякого конфлікту імен немає, оскільки однойменні змінні розташовані в різних просторах імен.

Приклад у файлах: example5.py та testex5.py

Скомпільовані файли Python

Кожний модуль Python може бути скомпільований і перетворений в особливу внутрішнє представлення (байт-код), – це робиться для прискорення виконання коду.

Файли з відкомпільованим кодом зберігаються в папці `__pycache__`, що автоматично створюється в папці, де перебуває сам файл з початковим, невідкомпільованим кодом модуля

Скомпільовані файли мають імена виду:

`<ім'я файлу з початковим кодом>.cpython-<перші дві цифри номера версії Python>.pyc`.

Одержання скомпільованого файлу

Приклад 6.

```
>>> import py_compile
>>> py_compile.compile('tests.py')
'__pycache__\\tests.cpython-35.pyc'
```

Таким чином, після запуску цього коду відкомпільований `tests.py` буде збережений у файлі

`tests.cpython-35.pyc`.

Імпортування скомпільованих файлів

Для імпортування модуля достатньо мати тільки файл з відкомпільованим кодом

Для прикладу перейменуємо файл `testcom.py` (наприклад, в `testcom.py`),

1.Скопіюємо файл `testcom.cpython-35.pyc` з папки `__pycache__` у папку з основною програмою

2. Перейменуємо його в `testcom35.pyc`.

3. Запустимо основну програму.

Програма буде нормально виконуватися.

Таким чином, щоб сховати початковий код модулів, можна поставляти програму клієнтам тільки з файлами, що мають розширення pyc.

Приклад у файлах:
usepyc.py, trstcom.py та testcom35.pyc

Порядок імпортування модулів

Імпортування модуля виконується тільки при першому виклику.

Інструкції

```
import <module>
from <module> import <function>
```

При кожному виклику інструкції `import` перевіряється наявність об'єкта модуля в словнику `modules` з модуля `sys`.

Якщо посилання на модуль перебуває в цьому словнику, то модуль повторно імпортуватися не буде.

Для прикладу виведемо ключі словника `modules`, попередньо відсортувавши їх.

Приклад 7.

```
import testcom35, sys# Підключаємо модулі tests і sys
print(sorted(sys.modules.keys()))
```

Результат виконання:

```
['__main__',    '_bootlocale',    '_codecs',
'_collections_abc',     '_frozen_importlib',
'_frozen_importlib_external', '_imp', '_io',
'_locale',    '_signal',    '_sitebuiltins',
'_stat', '_thread', '_warnings', '_weakref',
'_weakrefset', 'abc', 'builtins', 'codecs',
'encodings',              'encodings.aliases',
'encodings.cp1251',     'encodings.latin_1',
'encodings.mbcs',          'encodings.utf_8',
'errno',  'genericpath',  'io',  'marshal',
'nt',  'ntpath',  'os',  'os.path',  'site',
```

```
'stat', 'sys', 'sysconfig', 'testscom35',
'winreg', 'zipimport']
```

Динамічне імпортування модулів

Інструкція **import** вимагає явної вказівки об'єкта модуля.

Не можна передати назва модуля у вигляді рядка.

Щоб підключити модуль, назва якого створюється динамічно залежно від певних умов, слід скористатися функцією **__import__().**

Дпя прикладу підключимо модуль `tests.py` за допомогою функції `__import__()`

Приклад 8.

```
s = "test" + "s"  # динамічне створення назви модуля

m = __import__(s)  # Підключення модуля tests

print(m.x) # Вивід значення атрибута x

Результат виконання: 50

s = "math"  # динамічне створення назви модуля

m = __import__(s)  # Підключення модуля tests

print(m.pi) # Вивід значення атрибута x
```

```
Результат виконання: 3.141592653589793
```

Одержання списку ідентифікаторів модуля

Одержати список усіх ідентифікаторів усередині модуля дозволяє функція **dir().**

Крім того, можна скористатися словником **__dict__,** який містить усі ідентифікатори і їх значення.

Приклад 9.

```
import tests
print(dir(tests))
print(sorted(tests._dict_.keys()))
Результат виконання:
['__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', 'x']
```

['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'x']

Інструкція *from*

Для імпортування тільки певних ідентифікаторів з модуля можна скористатися інструкцією `from`.

Її формат такий:

```
from<Назв. модуля> import<Ідентифікатор1>
[as <Псевдонім1>]
[,...,<ІдентифікаторN> [as <Псевдонім N>]]
```

Дозволяє імпортувати модуль і зробити доступними тільки зазначені ідентифікатори.

Для довгих імен можна призначити псевдоніми, указавши їх після ключового слова `as`.

Виклик функції за псевдонімом

Як приклад зробимо доступними константу **`pi`** і функцію **`floor()`** з модуля **`math`**, а для назви функції створимо псевдонім.

Приклад 10. Інструкція from

```
from math import pi, floor as f
print(pi) # Вивід числа pi
# Викликаємо функцію floor() через ідентифікатор f
print(f(5.49))
```

```
Результат виконання:
3.141592653589793
5
```

Ідентифікатори в круглих дужках при багаторядковому запису

Ідентифікатори можна розмістити на декількох рядках, указавши їх назви через кому усередині круглих дужок:

Приклад 11.

```
from math import (pi, floor,
sin, cos)
print(pi) # Вивід числа pi
print(floor(5.49))
```

```
print(round(sin(2*pi),2))
print(cos(pi))
```

Огляд стандартної бібліотеки

Модулі стандартної бібліотеки можна умовно розбити на групи по тематиці.

1. **Сервіси періоду виконання**. Модулі: sys, atexit, copy, traceback, math, cmath, random, time, calendar, datetime, sets, array, struct, itertools, locale, gettext.
2. **Підтримка циклу розробки.** Модулі: pdb, hotshot, profile, unittest, pydoc. Пакети docutils, distutils.

3. **Взаємодія з ОС** (файли, процеси). Модулі: os, os.path, getopt, glob, popen2, shutil, select, signal, stat, tempfile.
4. **Обробка текстів**. Модулі: string, re, Stringio, codecs, difflib, mmap, sgmllib, htmllib, htmlentitydefs. Пакет xml.
5. **Багатопоточні обчислення.** Модулі: threading, thread, Queue.
6. **Зберігання даних. Архівація.** Модулі: pickle, shelve, anydbm, gdbm, gzip, zlib, zipfile, bz2, csv, tarfile.
7. **Платформо-залежні модулі**. Для UNIX: commands, pwd, grp, fcntl, resource, termios, readline, rlcompleter. Для Windows: msvcrt, _winreg, winsound.
8. **Підтримка мережі. Протоколи Інтернет**. Модулі: cgi, Cookie, urllib, urlparse, httplib, smtplib, poplib, telnetlib, socket, asyncore. Приклади серверів: Socketserver, Basehttpserver, xmlrpclib, asynchat.

9. **Підтримка Internet. Формати даних**. Модулі: quopri, uu, base64, binhex, binascii, rfc822, mimetools, Mimewriter, multifile, mailbox. Пакет email.
10. **Python про себе**. Модулі: parser, symbol, token, keyword, inspect, tokenize, pyclbr, py_compile, compileall, dis, compiler.
11. **Графічний інтерфейс**. Модуль Tkinter.

Сервіси періоду виконання

Модуль sys

Модуль sys містить інформацію про середовище виконання програми, про інтерпретатора Python. Далі будуть представлені найбільш популярні об'єкти із цього модуля: решта можна вивчити по документації.

exit([c])	Вихід із програми. Можна передати числовий код завершення: 0 у випадку успішного завершення, інші числа при аварійнім завершенні програми.
argv	Список аргументів командного рядка.

	Звичайно sys.argv[0] містить ім'я запущеної програми, а інші параметри передаються з командного рядка.
platform	Платформа, на якій працює інтерпретатор.
stdin, stdout, stderr	Стандартний ввід, вивід, вивід помилок. Відкриті файлові об'єкти.
version	Версія інтерпретатора.
setrecursionlimit(limit)	Установка рівня максимальної вкладеності рекурсивних викликів.
exc_info()	Інформація про оброблюване виключення.

Модуль time

Цей модуль дає функції для одержання поточного часу й перетворення форматів часу.

time.altzone - зсув DST часового поясу в секундах на захід від нульового меридіана. Якщо годинний пояс перебуває на схід, зсув негативний.

time.asctime([t]) - перетворює кортеж або struct_time у рядок виду "Thu Sep 27 16:42:37 2012". Якщо аргумент не зазначений, використовується поточний час.

time.clock() - в Unix, повертає поточний час. В Windows, повертає час, що пройшов з моменту першого виклику даної функції.

time.ctime([сек]) - перетворить час, виражений в секундах з початку епохи в рядок виду "Thu Sep 27 16:42:37 2012".

time.gmtime([сек]) - перетворить час, виражений в секундах з початку епохи в struct_time, де DST прапор завжди дорівнює нулю.

time.localtime([сек]) - як gmtime, але з DST прапором.

time.mktime(t) - перетворить кортеж або struct_time у число секунд із початку епохи. Обернена функції time.localtime.

time.sleep(сек) **-** призупинити виконання програми на задану кількість секунд.

time.strftime(формат, [t]) **-** перетворить кортеж або struct_time у рядок по формату:

ФОРМАТ	**ЗНАЧЕННЯ**
% a	Скрочена назва дня тижня
% A	Повна назва дня тижня
% b	Скорочена назва місяця
% B	Повна назва місяця
% c	Дата й час

% d	День місяця [01,31]
% H	Година (24-вартовий формат) [00,23]
% I	Година (12-вартовий формат) [01,12]
% j	День року [001,366]
% m	Номер місяця [01,12]
% M	Число хвилин [00,59]
% p	До полудня або після (при 12-годинному форматі)
% S	Число секунд [00,61]
% U	Номер тижня в році (нульовий тиждень починається з неділі) [00,53]
% w	Номер дня тижня [0(Sunday),6]
% W	Номер тижня в році (нульовий тиждень починається з понеділка) [00,53]
% x	Дата
% X	Час
% y	Рік без століття [00,99]

% Y	Рік зі століттям
% Z	Тимчасова зона
% %	Знак '%'

time.strptime(рядок [, формат]) **-** розбір рядка, що представляє час відповідно до формату. значення, що вертається, struct_time. Формат за замовчуванням: "%a %b %d %H:%M:%S %Y".

Клас **time.struct_time** - тип послідовності значення часу. Має інтерфейс кортежу. Можна звертатися по індексу або по імені.

1. tm_year
2. tm_mon
3. tm_mday
4. tm_hour
5. tm_min
6. tm_sec
7. tm_wday
8. tm_yday

9.tm_isdst

time.time() - час, виражене в секундах з початку епохи.

time.timezone - зсув місцевого годинного пояса в секундах на захід від нульового меридіана. Якщо годинний пояс перебуває на схід, зсув негативний.

time.tzname - кортеж із двох рядків: перша - ім'я DST вартового пояса, другий - ім'я місцевого годинного пояса.

Модулі array і struct

Ці модулі реалізують низькорівневий масив і структуру даних. Основне їхнє призначення - розбір двійкових форматів даних.

Модуль itertools

Цей модуль містить набір функцій для роботи з ітераторами. Ітератори дозволяють працювати з

даними послідовно, як ніби вони виходили в циклі. Альтернативний підхід - використання списків для зберігання проміжних результатів - вимагає часом великої кількості пам'яті, тоді як використання ітераторів дозволяє одержувати значення на момент, коли вони дійсно потрібні для подальших обчислень.

Модуль gettext

При інтернаціоналізації програми важливо не тільки передбачити можливість використання декількох культурних середовищ, але й переклад повідомлень і меню програми на відповідну мову. Модуль gettext дозволяє спростити цей процес досить стандартним способом. Основні повідомлення програми пишуться

англійською мовою. А переклади рядків, відзначених у програмі спеціальним образом, даються у вигляді окремих файлів, по одному на кожну мову (або культурне середовище). Уточнити нюанси використання gettext можна по документації до Python.