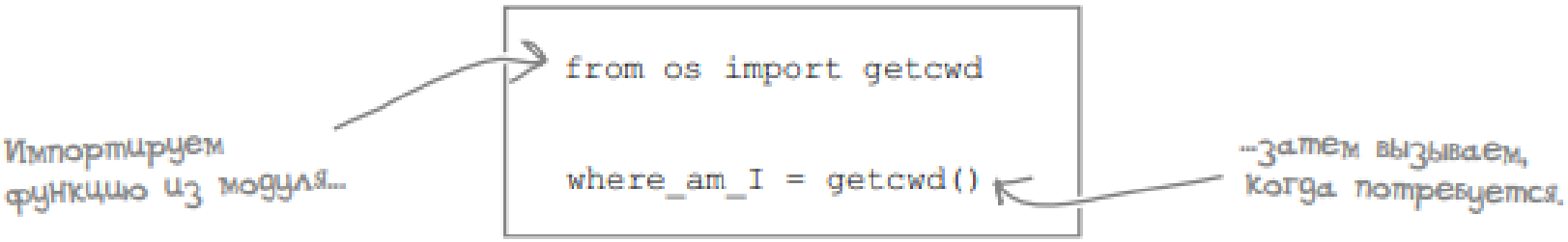
БИБЛИОТЕКА PYTHON

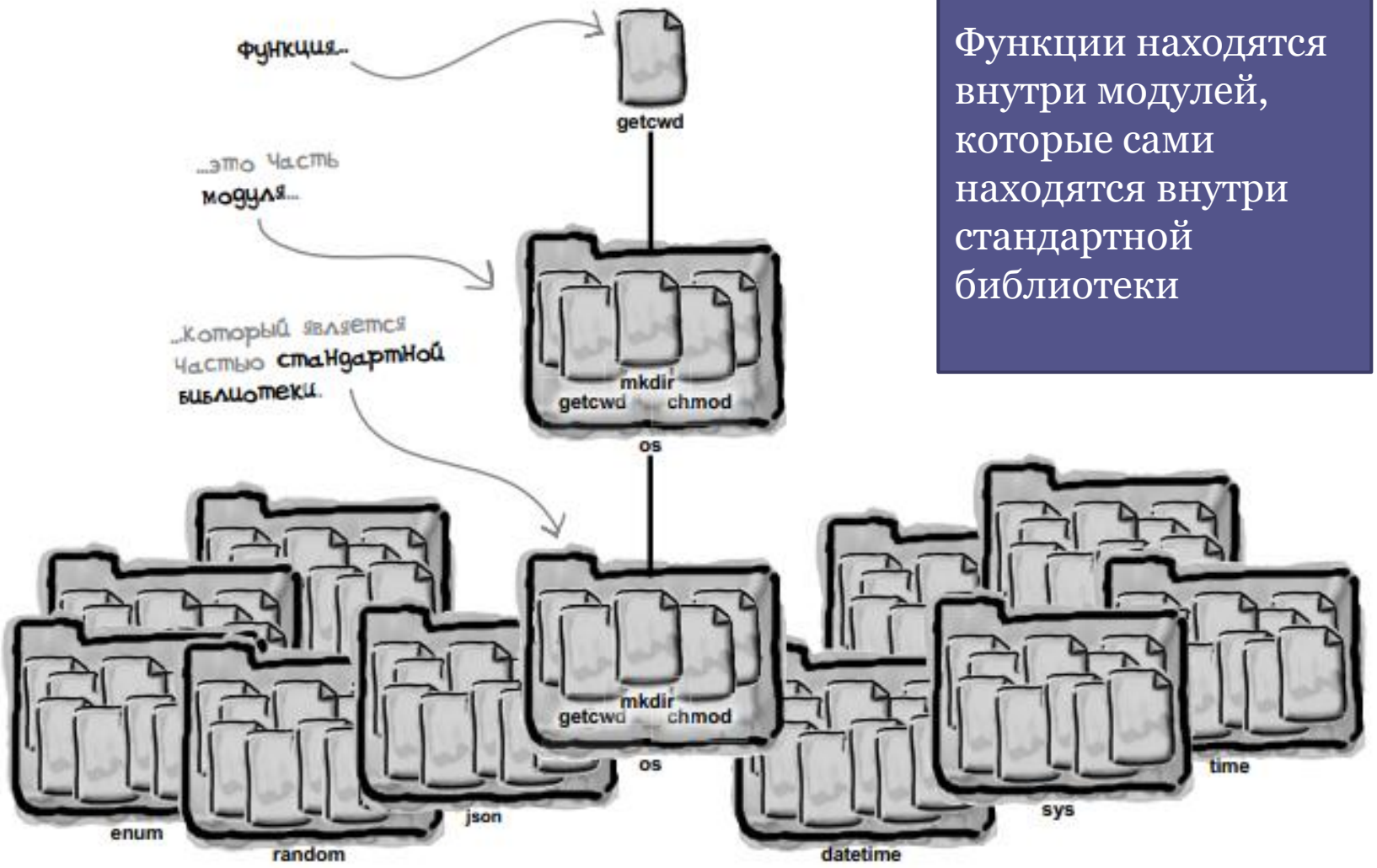
Стандартная библиотека Python очень богата и предоставляет много кода для повторного использования

Давайте посмотрим на другой модуль, под названием os, который предоставляет независимый от платформы способ взаимодействия с операционной системой. Рассмотрим только одну из его функций: getcwd, которая при вызове возвращает текущий рабочий каталог. Вот типичный пример импорта и вызова этой функции в программе на Python:



Набор связанных функций составляет программный модуль, и в стандартной библиотеке есть много модулей:

Функции находятся внутри модулей, которые сами находятся внутри стандартной библиотеки



Стандартная библиотека — это бриллиант в короне Python, она содержит модули на все случаи жизни, которые помогут, например, упаковать или распаковать ZIP-архив, отправить или принять электронную почту, обработать HTML-страницу. Стандартная библиотека даже включает веб-сервер, а также механизм для работы с популярными базами данных SQLite.

Хотя Python гордится своей кросс-платформенностью, то есть код, написанный на одной платформе, может выполняться (как правило, без изменения) на другой, иногда важно знать, что вы работаете, скажем, в Mac OS X. Больше узнать о системе вам поможет модуль **sys**. Чтобы определить операционную систему, сначала импортируйте модуль **sys,** а затем обратитесь к атрибуту **platform**:

```
>>> import sys
>>> sys.platform
'darwin'
```

Поэкспериментируем в darwin (ядро Mac OS X).
Импортируем нужный модуль и обращаемся к атрибуту.

Модуль **sys** — это хороший пример повторно используемого модуля, который в первую очередь предоставляет доступ к предустановленным атрибутам (таким как platform). В следующем примере мы определим, какая версия Python используется, и передадим это значение функции print для вывода на экран:

```
>>> print(sys.version)
3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
```

Информация об используемой версии Python.

Модуль **os** — это хороший пример повторно используемого модуля, который в первую очередь предоставляет доступ к богатому набору функций и реализует универсальный способ взаимодействия кода на Python с операционной системой, независимо от ее вида.

Ниже показано, как получить имя папки, в контексте которой выполняется код, используя функцию **getcwd**. Перед вызовом функции импортируем модуль:

```
>>> import os
>>> os.getcwd()
'/Users/HeadFirst/CodeExamples'
```

Импортируем модуль, затем вызываем нужную функцию.

Работать с датами (и временем) приходится много, и стандартная библиотека предоставляет модуль **datetime** чтобы помочь вам, когда вы работаете с этим типом данных. Функция date.today возвращает текущую дату:

```
>>> import datetime
>>> datetime.date.today()
datetime.date(2015, 5, 31)
```

Текущая дата.

Странный способ отображения текущей даты, не находите? Отдельно число, месяц и год можно получить, добавляя соответствующий атрибут доступа к вызову date.today:

```
>>> datetime.date.today().day
31
>>> datetime.date.today().month
5
>>> datetime.date.today().year
2015
```

Части, составляющие текущую дату.

Также можно вызвать функцию date.isoformat и передать ей текущую дату, чтобы получить более удобочитаемую версию текущей даты, которая получается преобразованием даты в строку с помощью isoformat:

```
>>> datetime.date.isoformat(datetime.date.today())
'2015-05-31'
```

Текущая дата в виде строки

Теперь перейдем к времени, которого, кажется, всем нам не хватает. Может ли стандартная библиотека сказать, который сейчас час? Да. Для этого, после импортирования модуля **time**, надо вызвать функцию strftime, определив формат отображения времени. Например, если вам интересно узнать текущее время в часах (%I) и минутах (%M) в 12-часовом формате:

```
>>> import time
>>> time.strftime("%I:%M")
'11:55'
```

Боже мой! Это время?

А как узнать день недели и определить текущую половину суток — до полудня или после? В этом вам помогут спецификаторы %A%p:

```
>>> time.strftime("%A %p")
'Sunday PM'
```

Мы выяснили, что сейчас воскресенье, без пяти минут полночь... возможно, пора спать?

В качестве примера функциональных возможностей стандартной библиотеки представьте, что у вас есть некоторые HTML документы, и вы беспокоитесь, что они могут содержать потенциально опасные теги

```
>>> import html
>>> html.escape("This HTML fragment contains a <script>script</script> tag.")
'This HTML fragment contains a &lt;script&gt;script&lt;/script&gt; tag.'
>>> html.unescape("I &hearts; Python's &lt;standard library&gt;.")
"I ♥ Python's <standard library>."
```

Преобразование разметки HTML в экранированный текст и обратно

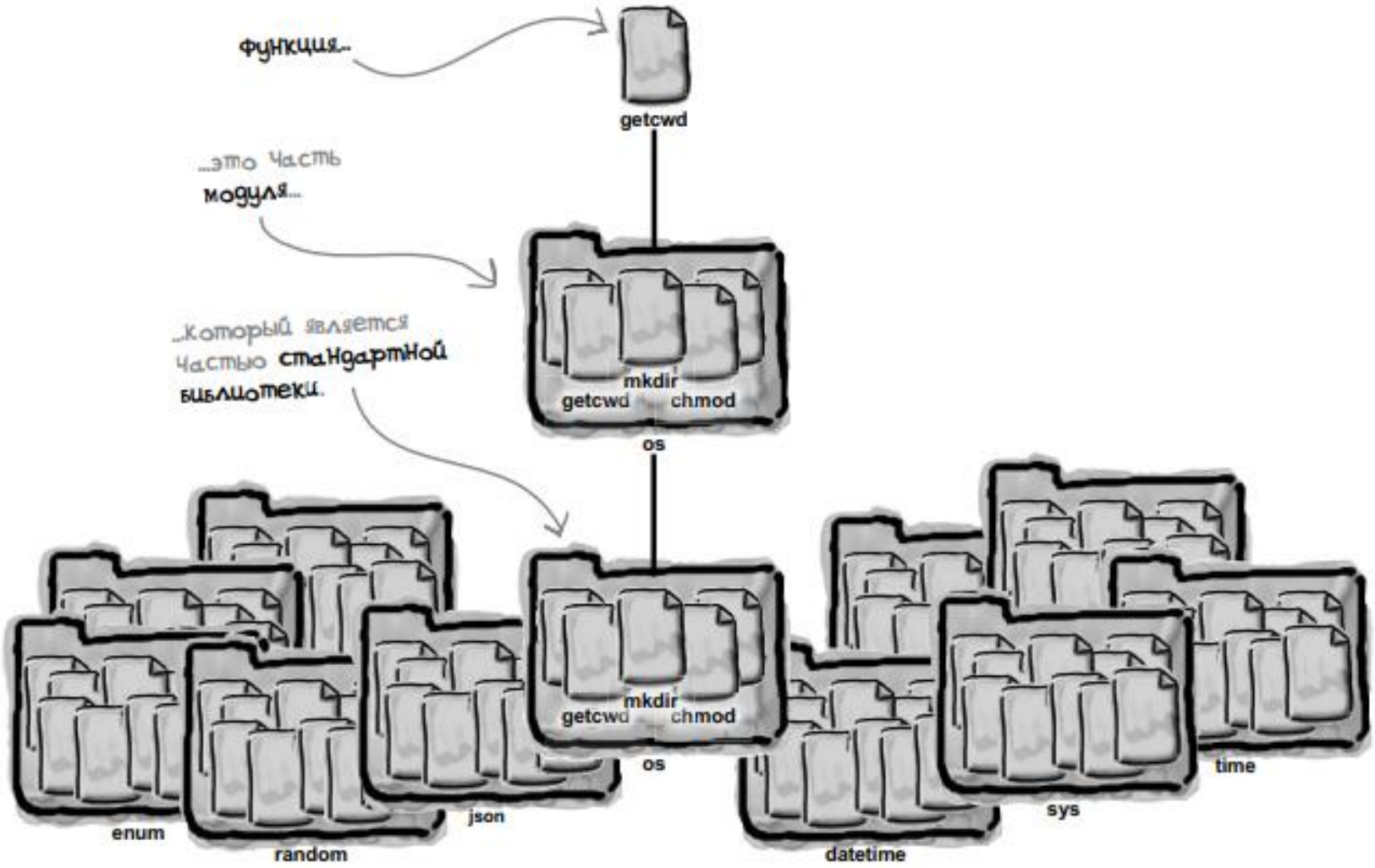
комплекте

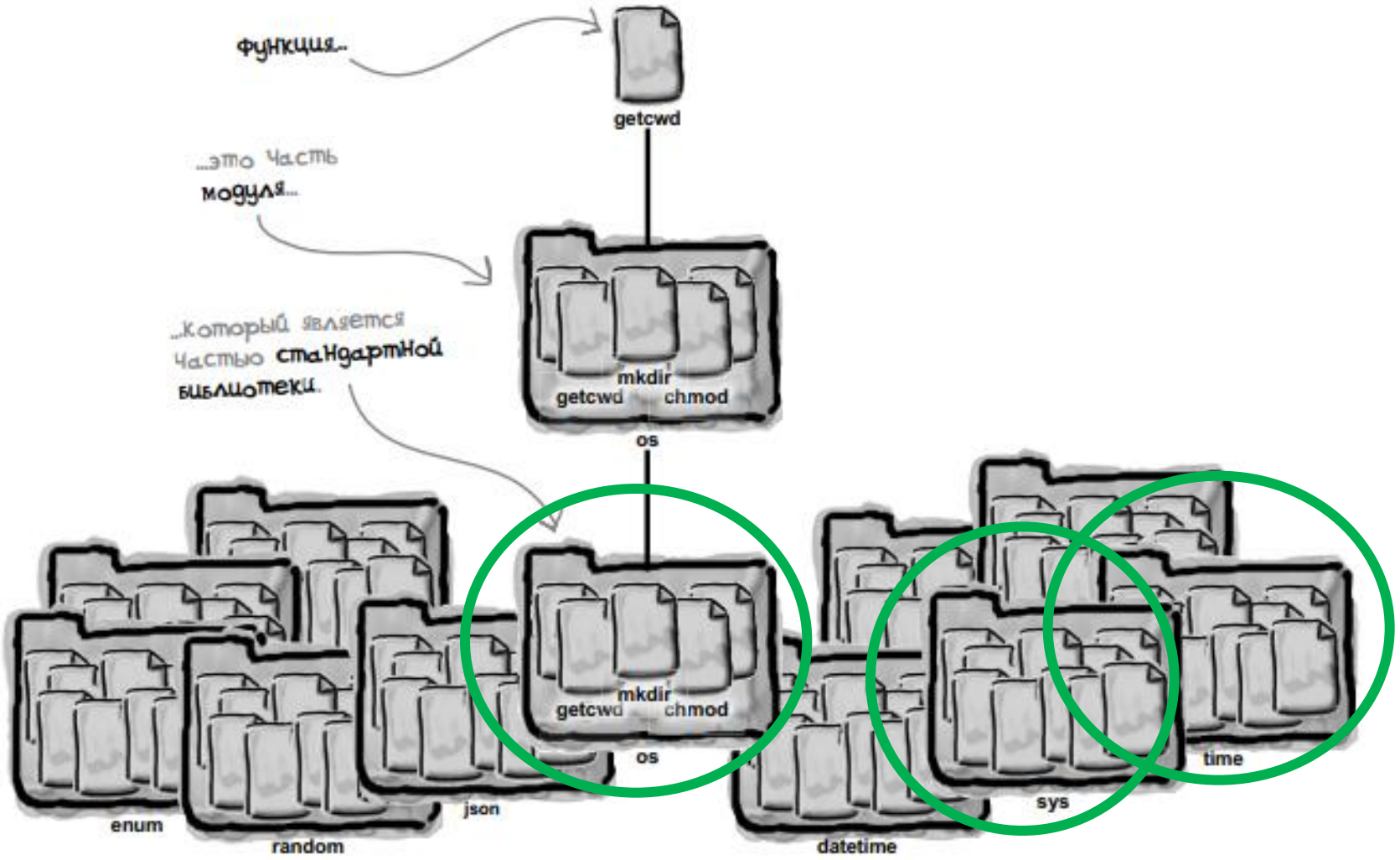
Стандартная библиотека настолько богата, что мышление — это все, что вам нужно, чтобы немедленно приступить к работе с языком Python, сразу после его установки. В отличие от рождественского утра, когда вы открываете новую игрушку и обнаруживаете, что она поставляется без батареек, Python вас не разочарует; он поставляется со всем необходимым для начала работы. Это относится не только к модулям стандартной библиотеки: не забывайте о наличии в комплекте IDLE — маленькой, но достаточно удобной IDE. От вас требуется только начать писать ко

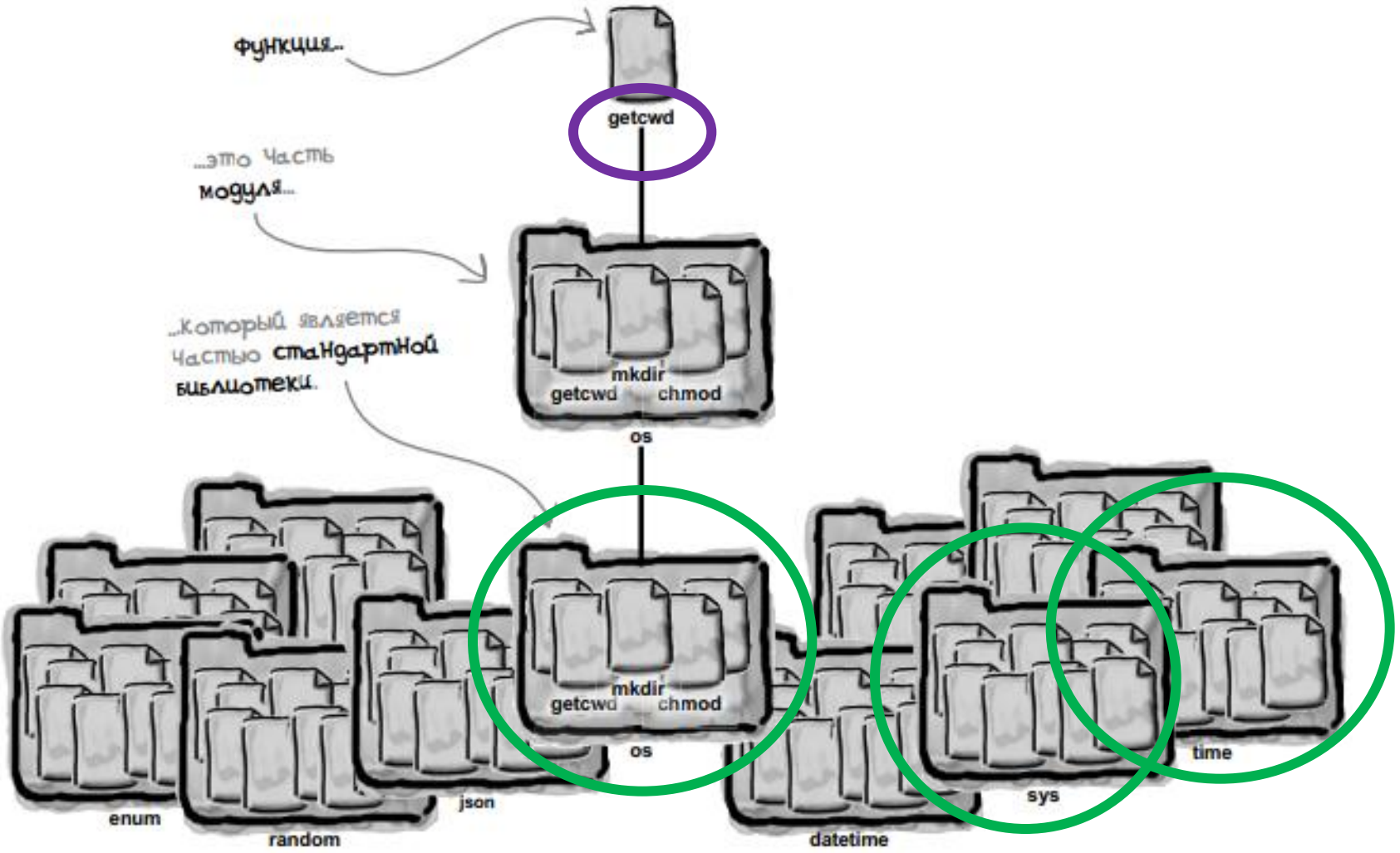
Как именно узнать, что именно делает конкретный модуль стандартной библиотеки?☐

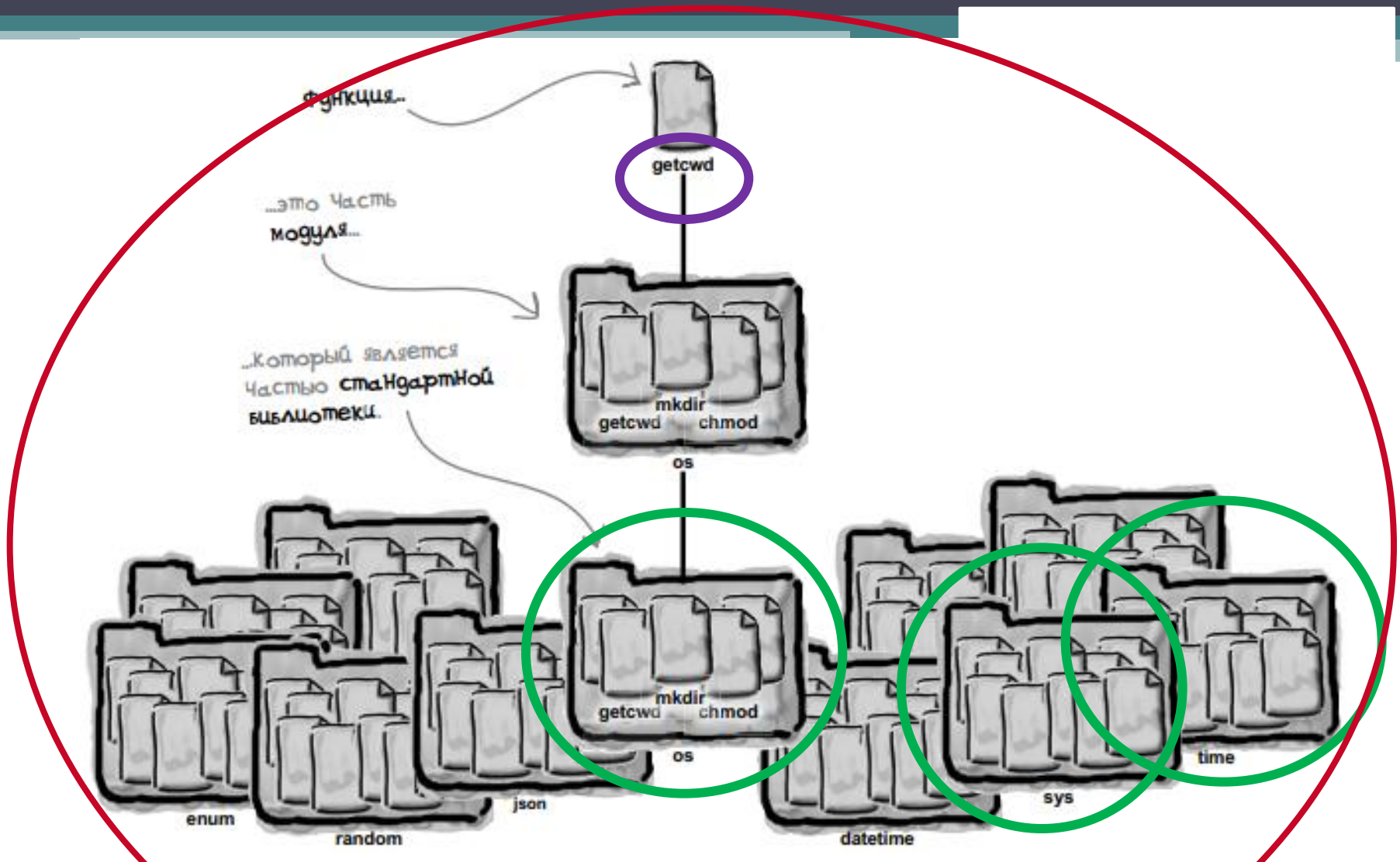
Документация Python содержит все ответы на вопросы о стандартной библиотеке. Вот отправная точка
☐*https://docs. python.org/3/library/index.html*

Стандартная библиотека — это не единственное место, где можно найти отличные модули для использования в своих программах. Сообщество Python поддерживает обширную коллекцию сторонних модулей, некоторые из них рассмотрены далее. Если вы хотите предварительно с ними ознакомиться, зайдите в репозиторий сообщества по адресу *http://pypi.python.org*


функция...
getcwd
...это часть модуля...
mkdir
getcwd
chmod
os
...который является частью стандартной библиотеки.
mkdir
getcwd
chmod
os
enum
random
json
datetime
sys
time


функция...
getcwd
...это часть модуля...
mkdir
getcwd
chmod
os
...который является частью стандартной библиотеки.
mkdir
getcwd
chmod
os
enum
random
json
datetime
sys
time


функция...
getcwd
...это часть модуля...
mkdir
getcwd
chmod
os
...который является частью стандартной библиотеки.
mkdir
getcwd
chmod
os
enum
random
json
datetime
sys
time



Функции + модули = стандартная библиотека

Модули Python для Maya

Сценарии Python можно использовать для многих задач в Maya, от выполнения простых команд до разработки подключаемых модулей, и доступно несколько различных библиотек, связанных с Maya, предназначенных для различных задач. Ниже приводится краткий обзор библиотек Python, которые сотрудничают с Maya:

maya.cmds

pymel.core

maya.OpenMaya

maya.api.OpenMaya

maya.cmds
Это оболочка Python для команд MEL, которую можно использовать вместо MEL.

pymel.core
Pymel - это альтернативная оболочка для MEL, разработанная третьей стороной. Он поставляется с Maya, но не поддерживается Autodesk. Он по-другому организует команды и использует объектно-ориентированный подход по сравнению с процедурным подходом maya.cmds.

maya.OpenMaya
Это оболочка Python для Maya C ++ API, называемая Python API 1.0. Он подходит для разработки подключаемых модулей и других задач, требующих функциональности, не предоставляемой MEL.

maya.api.OpenMaya
Это оболочка Python для Maya C ++ API, называемая Python API 2.0. Эта оболочка имеет лучшую производительность и более «питонична», чем Python API 1.0. Это также более новый API, который все еще находится в стадии разработки, поэтому не все классы, представленные в 1.0, доступны.

Использование maya.cmds

Autodesk Maya поддерживает использование сценариев в стиле Python везде, где вы использовали команды MEL. Реализация сценариев Python в Maya обеспечивает тот же доступ к собственным командам Maya, что и через MEL. То есть все встроенные команды Maya, такие как сфера, ls и т. Д., Доступны через Python.

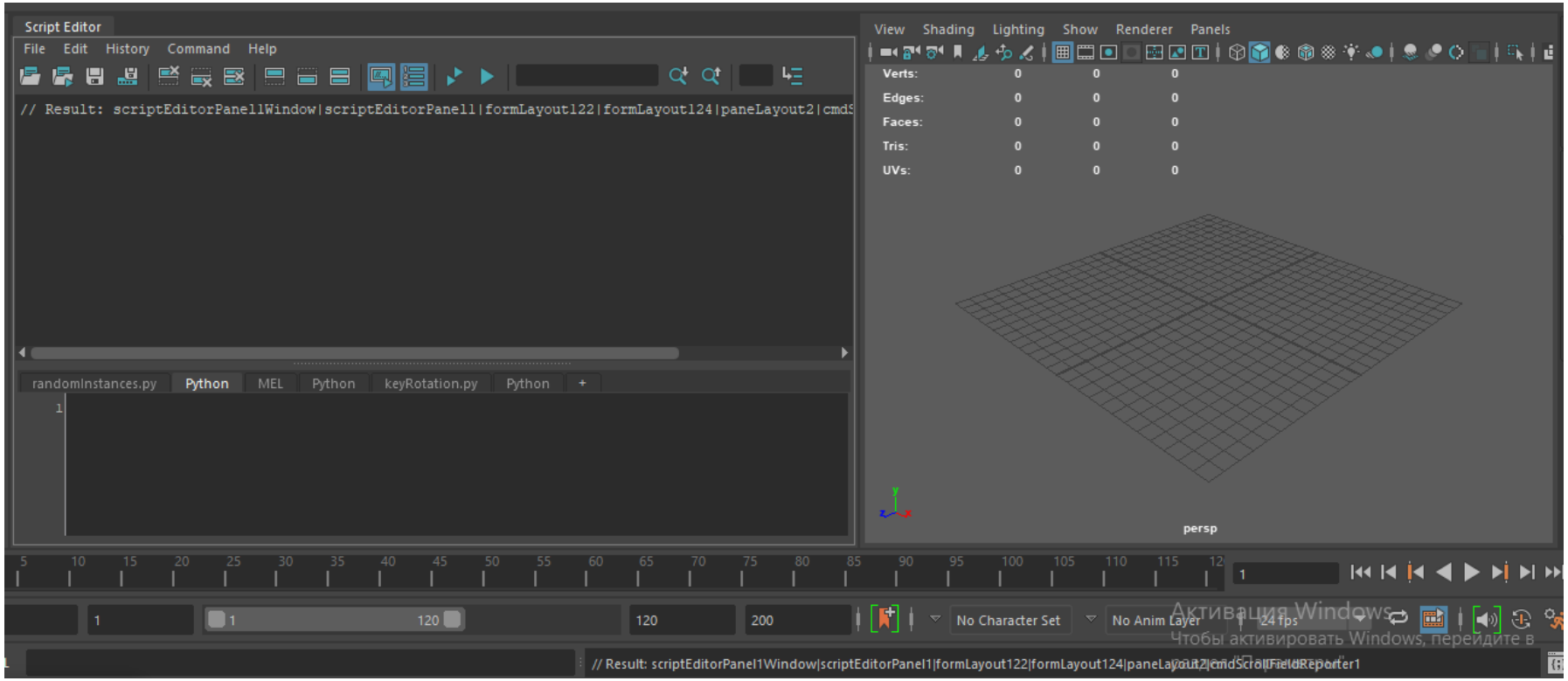
Некоторые встроенные модули MEL недоступны в Python, хотя обычно у них есть аналоги на Python. К ним относятся математические функции (abs, sin, cos, ...) и строковые функции (match, gmatch, tokenize ...). (Набор встроенных функций MEL можно рассматривать как библиотеку времени выполнения MEL.) Python поставляется с широким спектром стандартных модулей, обеспечивающих схожую функциональность.

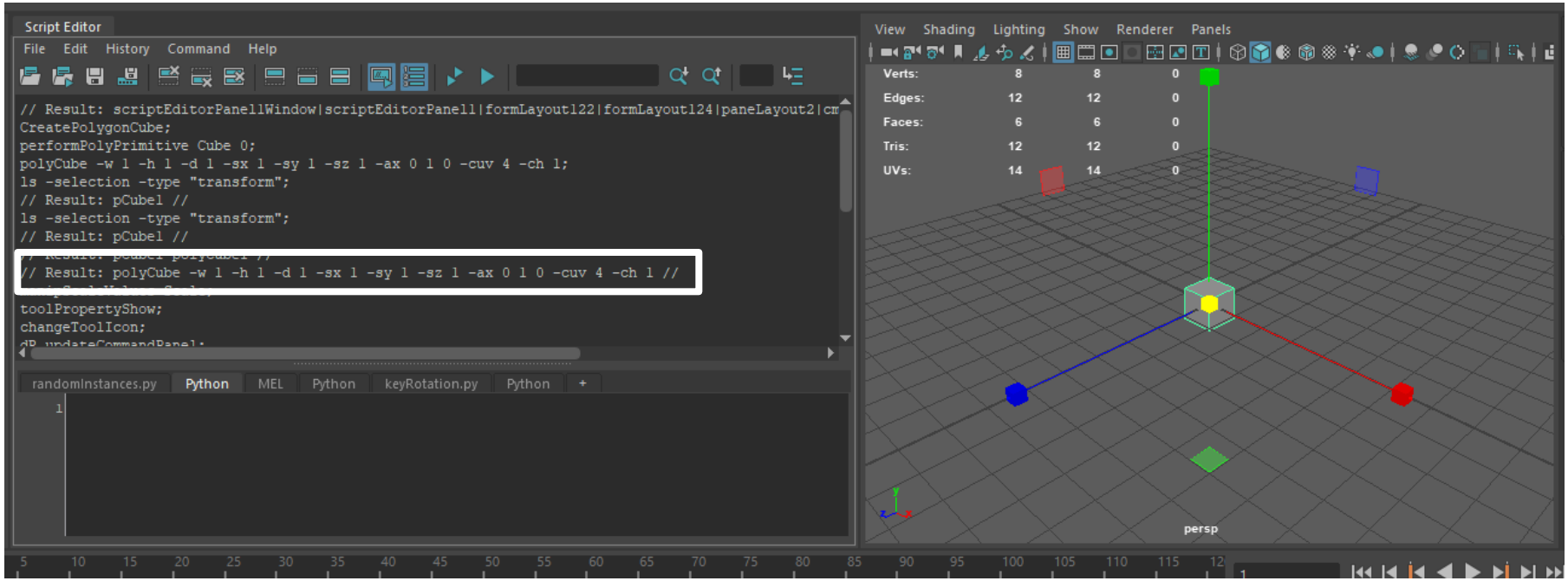
```
import maya.cmds as cmds
```

Геометрия

Разбираем геометрию по запчастям и собираем обратно.

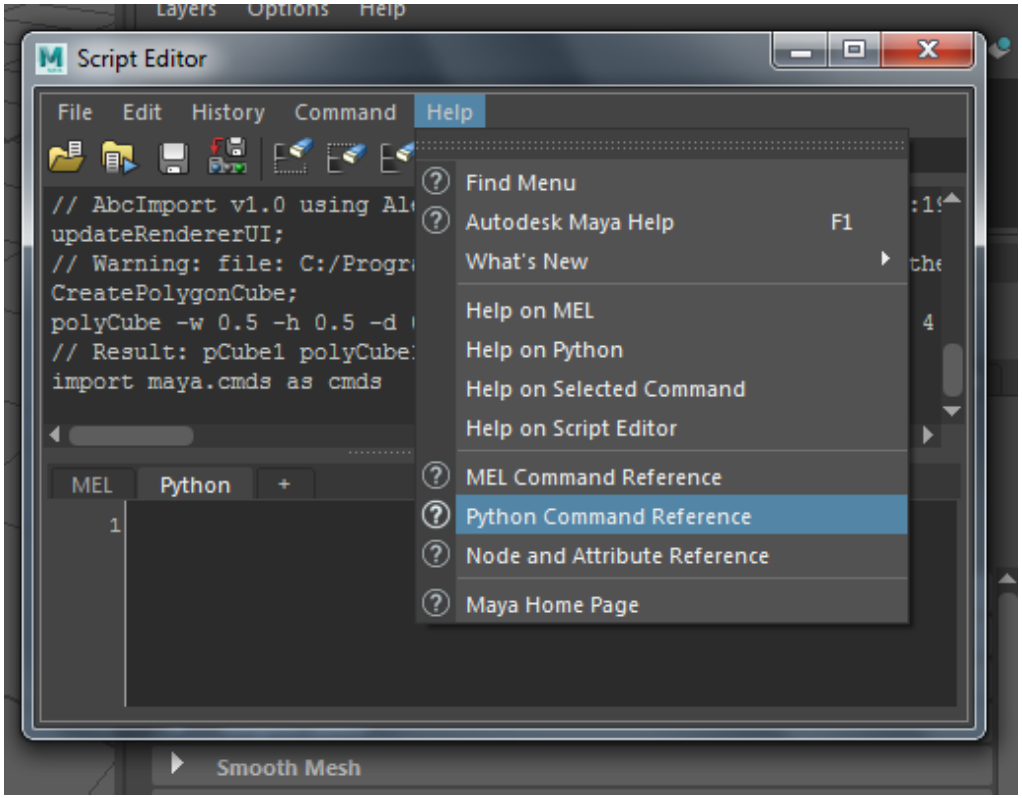
Установимо Script Editor у Layout, клацнувши правою кнопкою миші на піктограму макета.



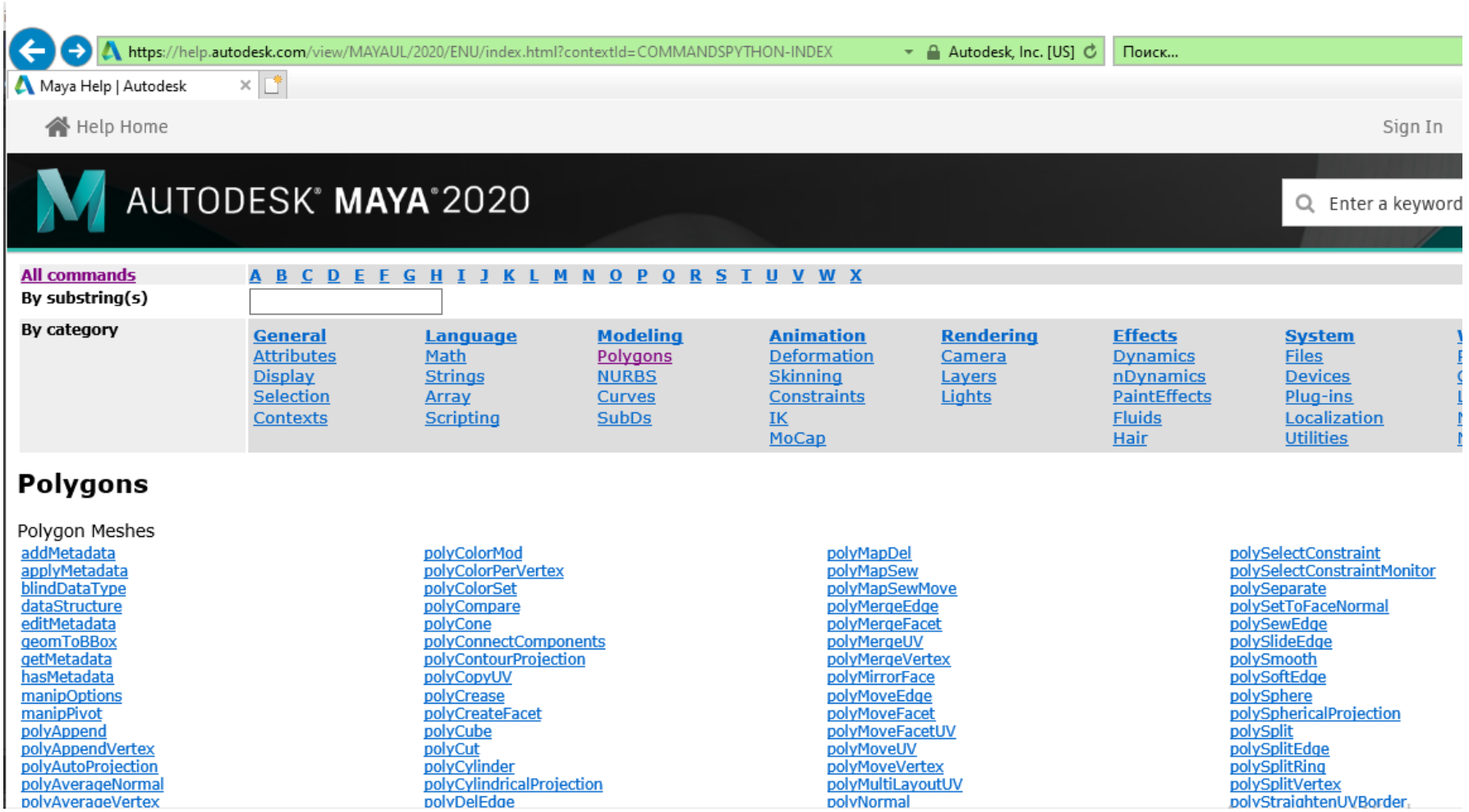


CreatePolygonCube;
polyCube -w 0.5 -h 0.5 -d 0.5 -sx 1 -sy 1 -sz 1 -ax 0 1 0 -cuv 4 -ch 1;
// Result: pCube1 polyCube1 //

Щоб запустити цю команду в Python, виберіть вкладку Python 'import maya.cmds як cmds'. Цей модуль надає більшість команд MEL для Python. Щоб дізнатись, які команди доступні, виберіть Help - Python Command Reference. Веб-браузер повинен відкрити список алфавітно відсортованих команд.



Шукайте "PolyCube". Коли наш куб був створений, команда "MEL", що повторювала "PolyCube", отримала такі параметри.



command (Python)

polyCube

MEL version

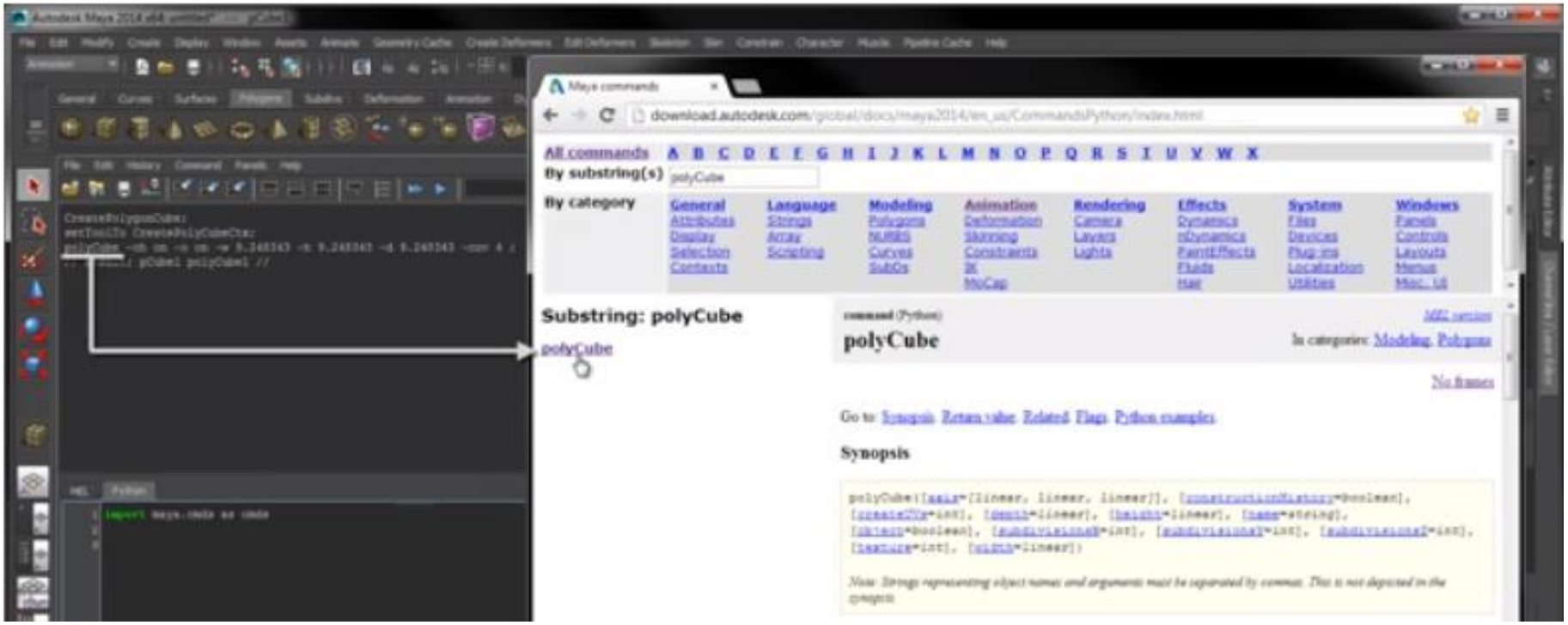
In categories: Modeling, Polygons

No frames

Go to: Synopsis. Return value. Related. Flags. Python examples.

Synopsis

```
polyCube([axis=[linear, linear, linear]], [caching=boolean], [constructionHistory=boolean], [createUVs=int],
[depth=linear], [height=linear], [name=string], [nodeState=int], [object=boolean], [subdivisionsDepth=int],
[subdivisionsHeight=int], [subdivisionsWidth=int], [subdivisionsX=int], [subdivisionsY=int], [subdivisionsZ=int],
[texture=int], [width=linear])
```



polyCube -w 0.5 -h 0.5 -d 0.5 -sx 1 -sy 1 -sz 1 -ax 0 1 0 -cuv 4 -ch 1;

Літери "**w**", "**h**" та "**d**" відповідно позначають значення ширини, висоти та глибини куба.

Літери “**sx**", “**sy**" та “**sz**" відповідно позначають кількість полігонів продовж вісі (subdivisions).

ax - вісі

cuv - Create UVs or not.0: No UVs
1: No Normalization
2: Normalize Each Face Separately
3: Normalize Collectively
4: Normalize and Preserve Aspect Ratio

Параметр "**ch**" представляє історію будівництва.

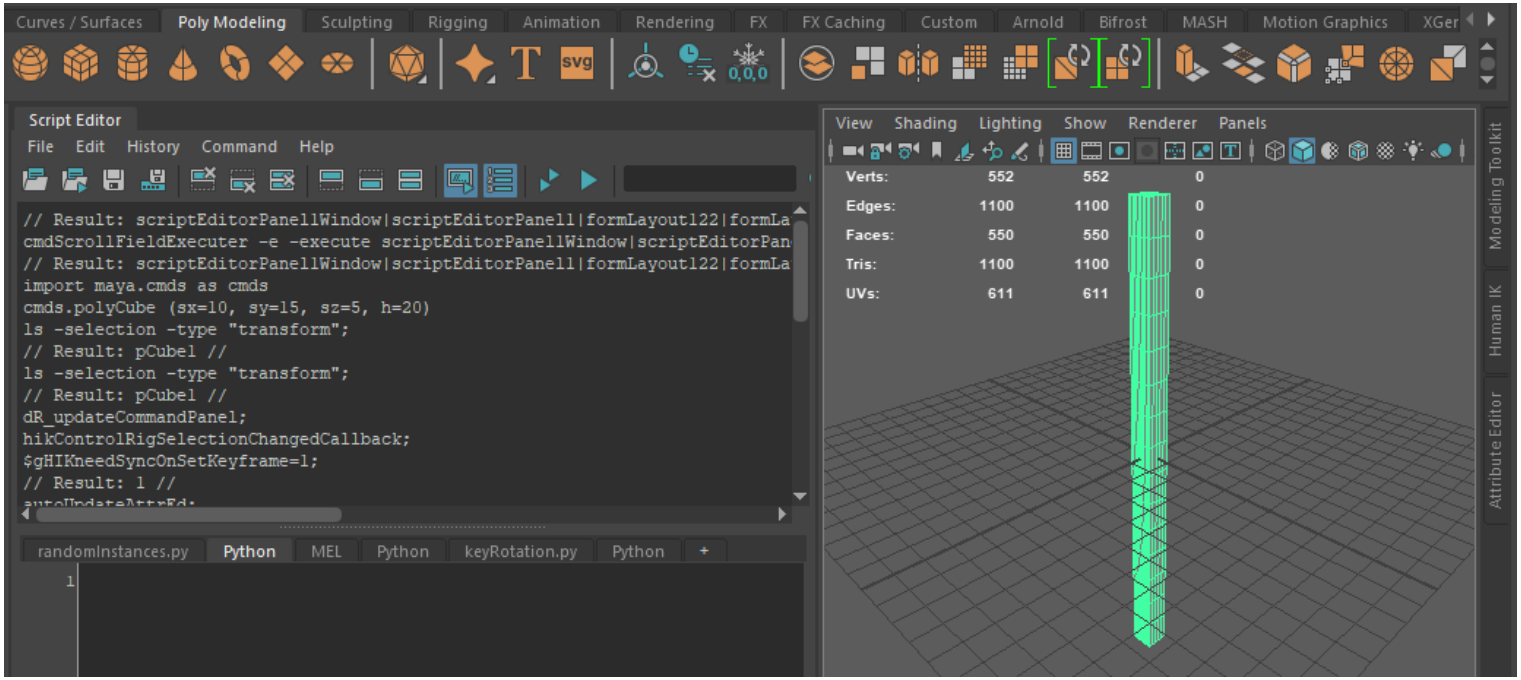
Зверніть увагу, що ключові слова " **on** " та " **off** " в MEL переводяться до " **True** " та " **False** " ключових слів у Python.

Параметр '**o**' або '**object**' створює об'єкт у сцені.

Приклад

Щоб безпосередньо перекласти команду MEL на Python, введіть такий рядок.

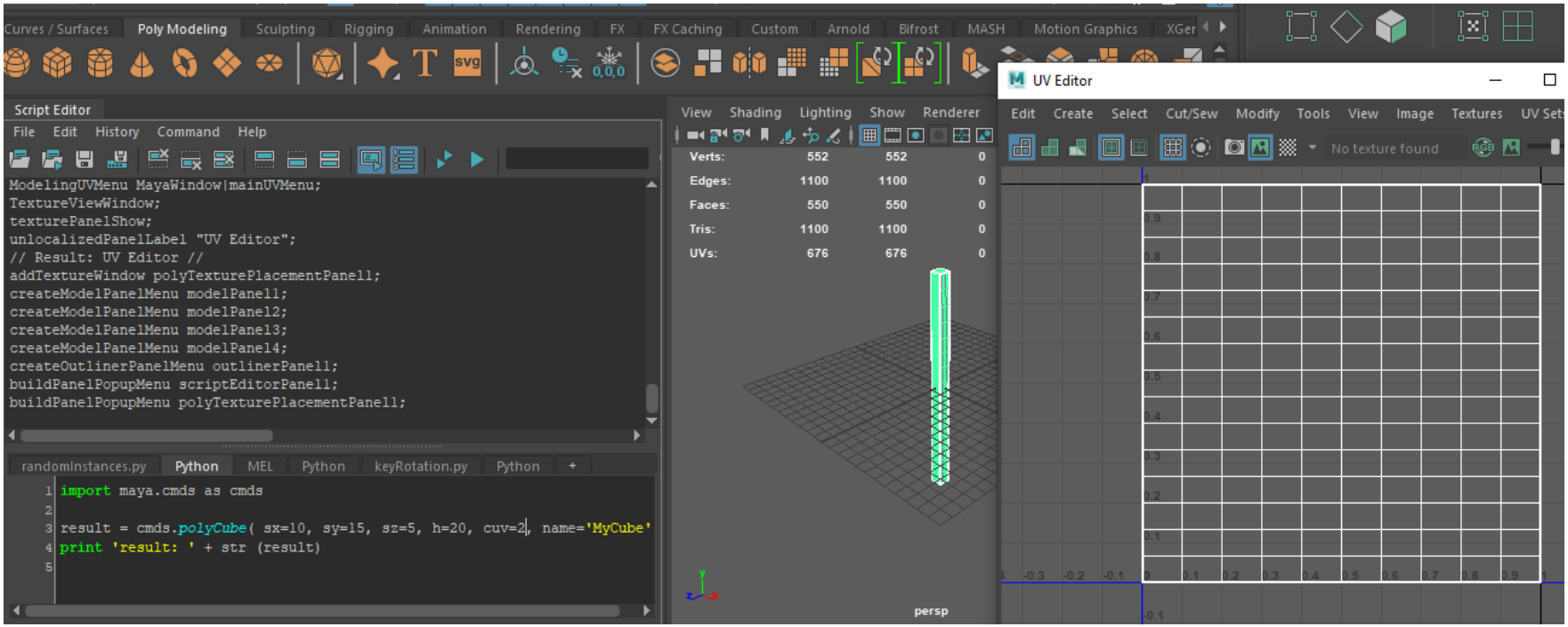
```
import maya.cmds as cmds

cmds.polyCube( sx=10, sy=15, sz=5, h=20 )
#result is a 20 units height rectangular box
#with 10 subdivisions along X, 15 along Y and 5 along Z.
```



Приклад (UV)

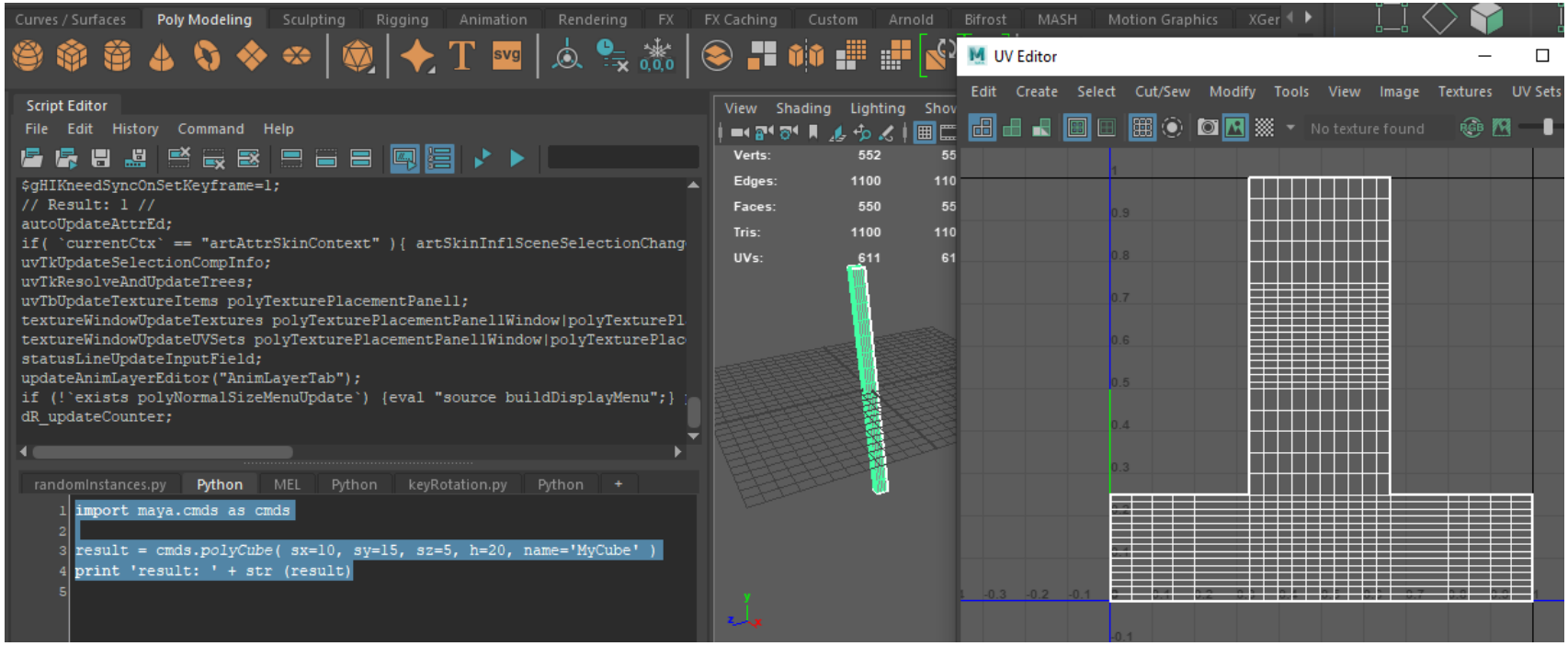
Задаємо cuv, що дорівнює 2 (Normalize Each Face Separately)

```
import maya.cmds as cmds
cmds.polyCube( sx=10, sy=15, sz=5, h=20, cuv=2 )
```



Якщо додатково не вказувати uv, автоматично у об'єкта будуть стандартні uv

```
import maya.cmds as cmds
cmds.polyCube( sx=10, sy=15, sz=5, h=20)
```



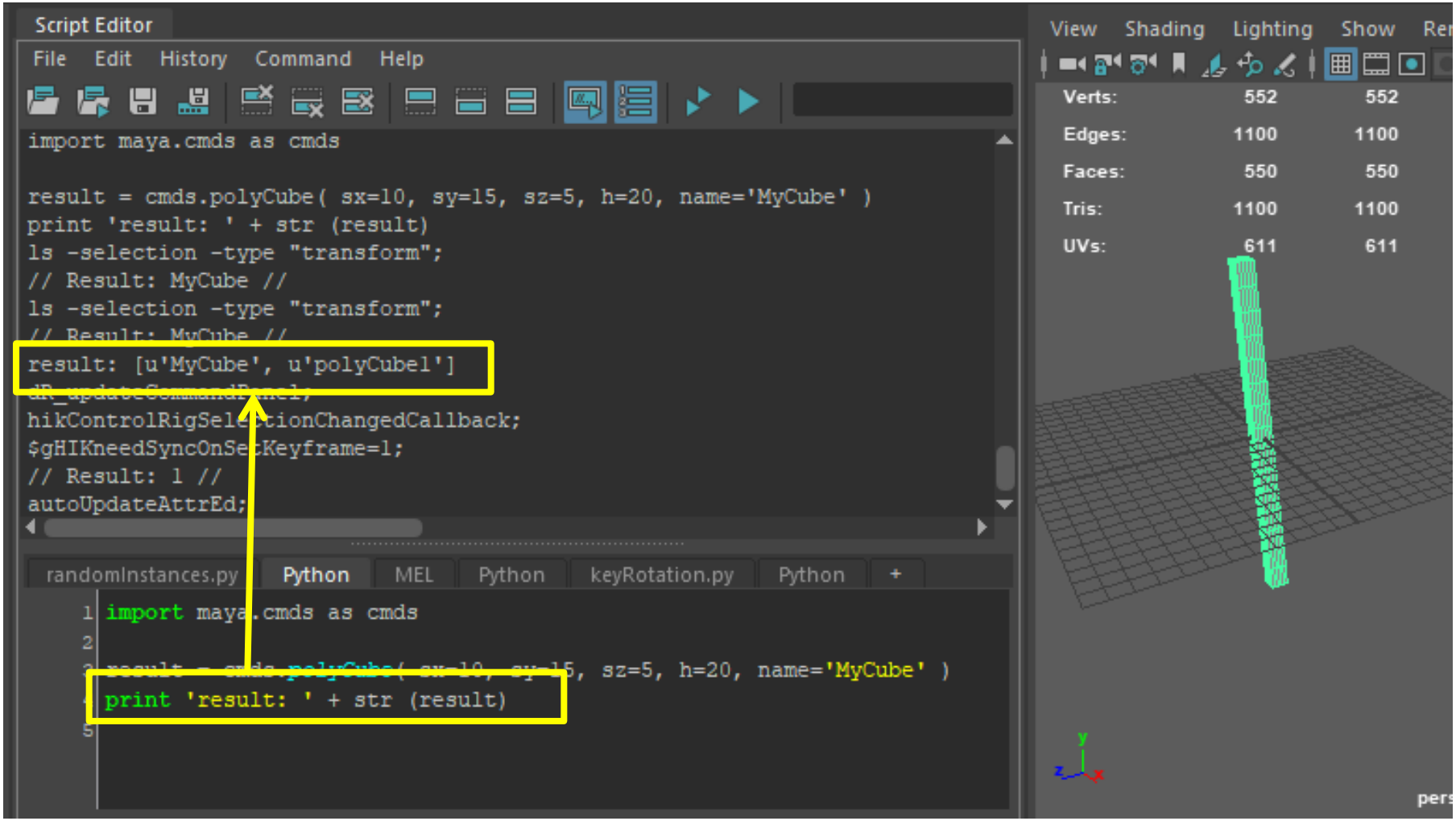
Налаштуємо назву цього куба, додайте параметр імені та встановіть його на "myCube“

```
import maya.cmds as cmds

result = cmds.polyCube( sx=10, sy=15, sz=5, h=20, name='MyCube' )
```

На сцені створюється новий куб.
Запустіть таку версію друку, щоб побачити, що Майя повертає у змінній "результат".
Функція 'str' перетворює зміст змісту в звичайний текст.

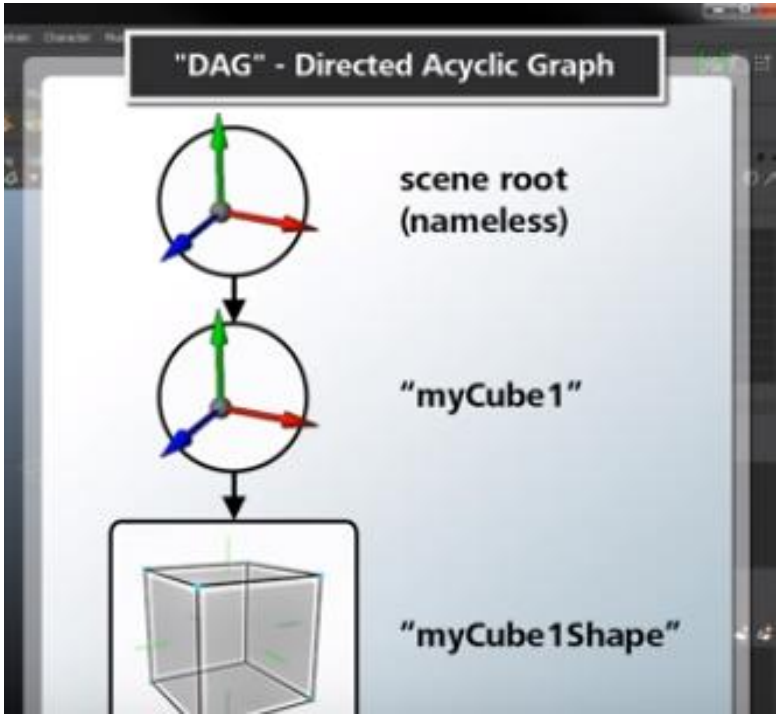
```
print 'result: ' + str (result)
```

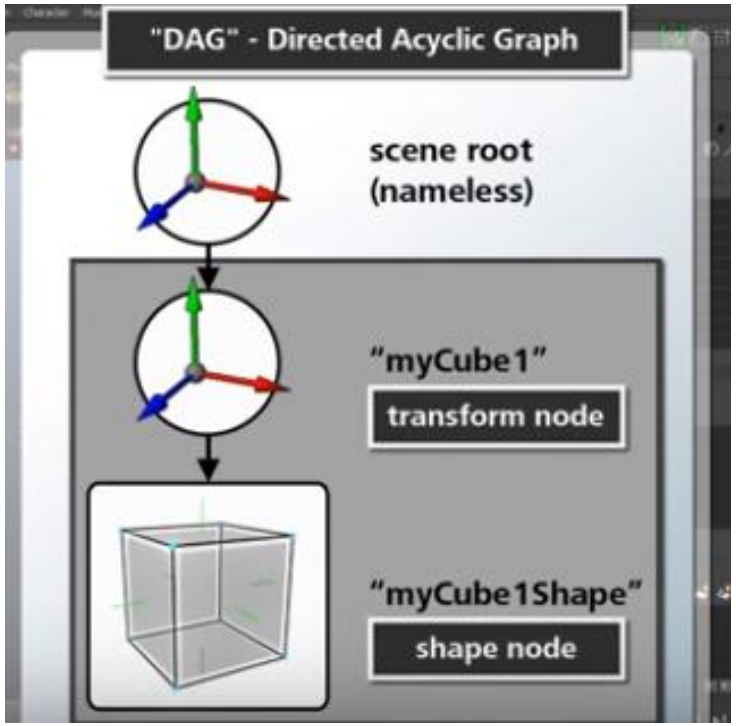
На панелі виводу ми бачимо, щозмінна 'result' фактично вказує на список з двох.



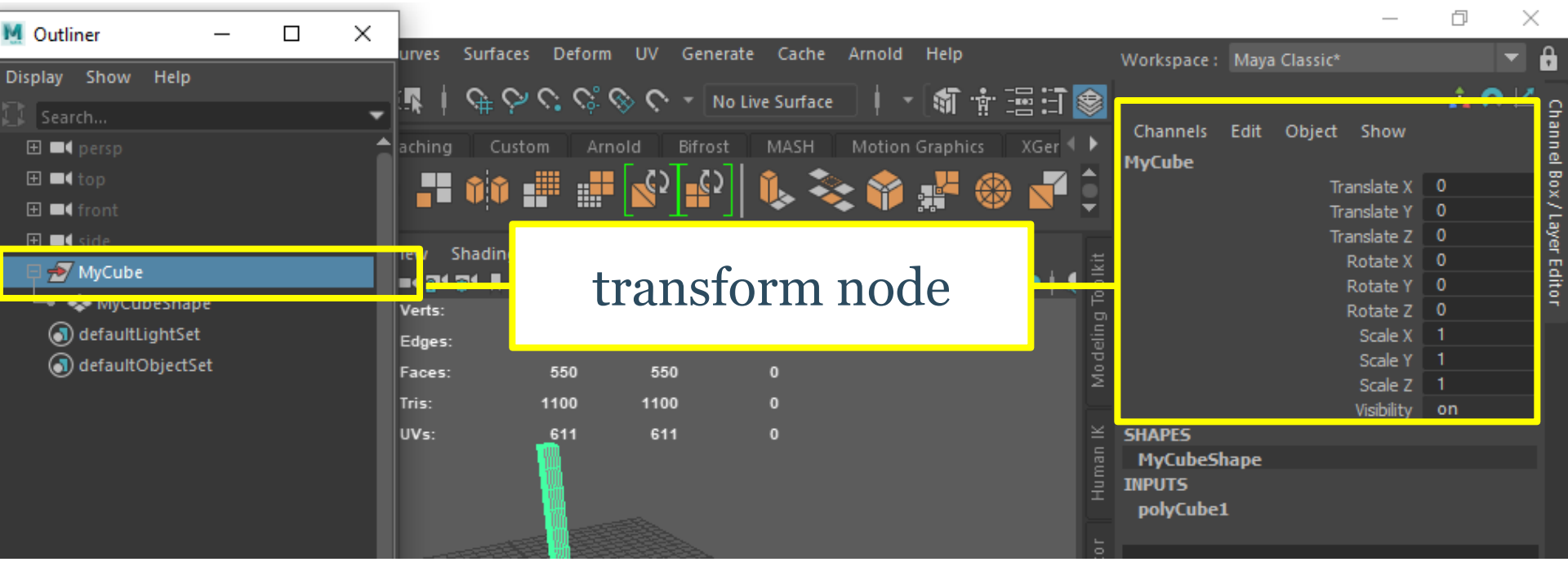
Першим елементом, ‘**MyCube**', є назва вузла перетворення куба. Другийелемент, '**polyCube**', - це назва структури даних, що лежить в основі куба. У нижньому регістрі "u" вказується, що рядок закодований у форматі текстового формату Unicode.

Щоб зрозуміти, на що посилаються **Mycube** та **polyCube**, зверніть увагу, що сцена майя організована як графік, який часто називають "DAG” (Directed Acyclic Graph).

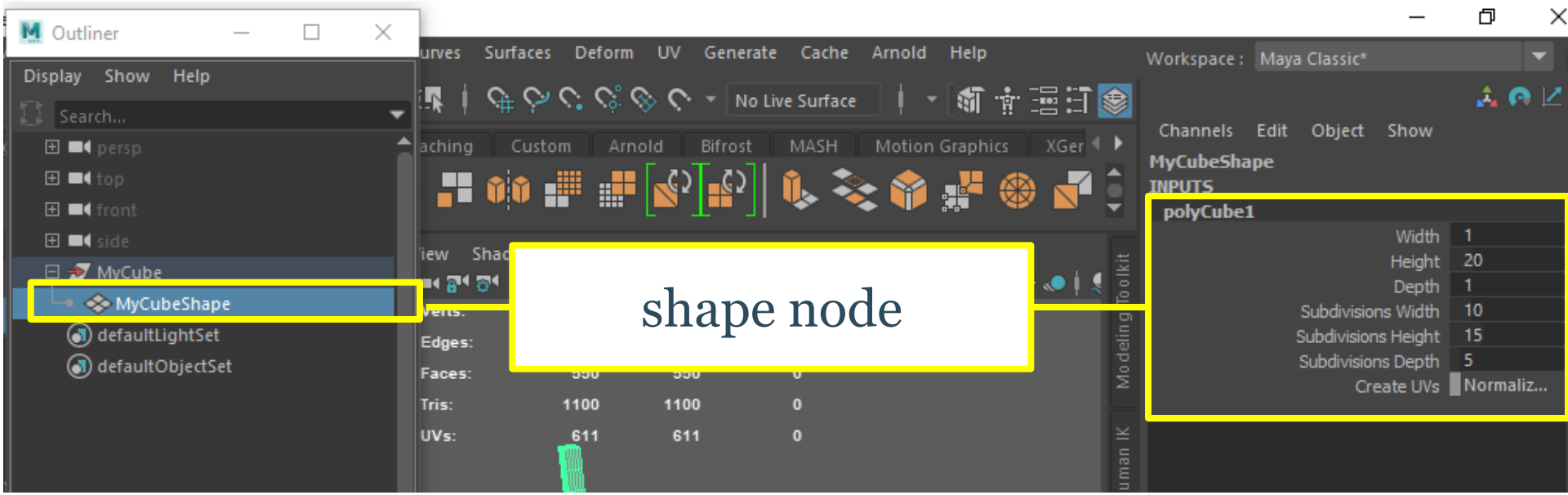




Щоб отримати чітке поняття про ці відносини, відкрийте Outliner.



MyCube – це вузол перетворення, який визначає, де в просторі існує куб. Це включає його переклад, обертання та масштаб.



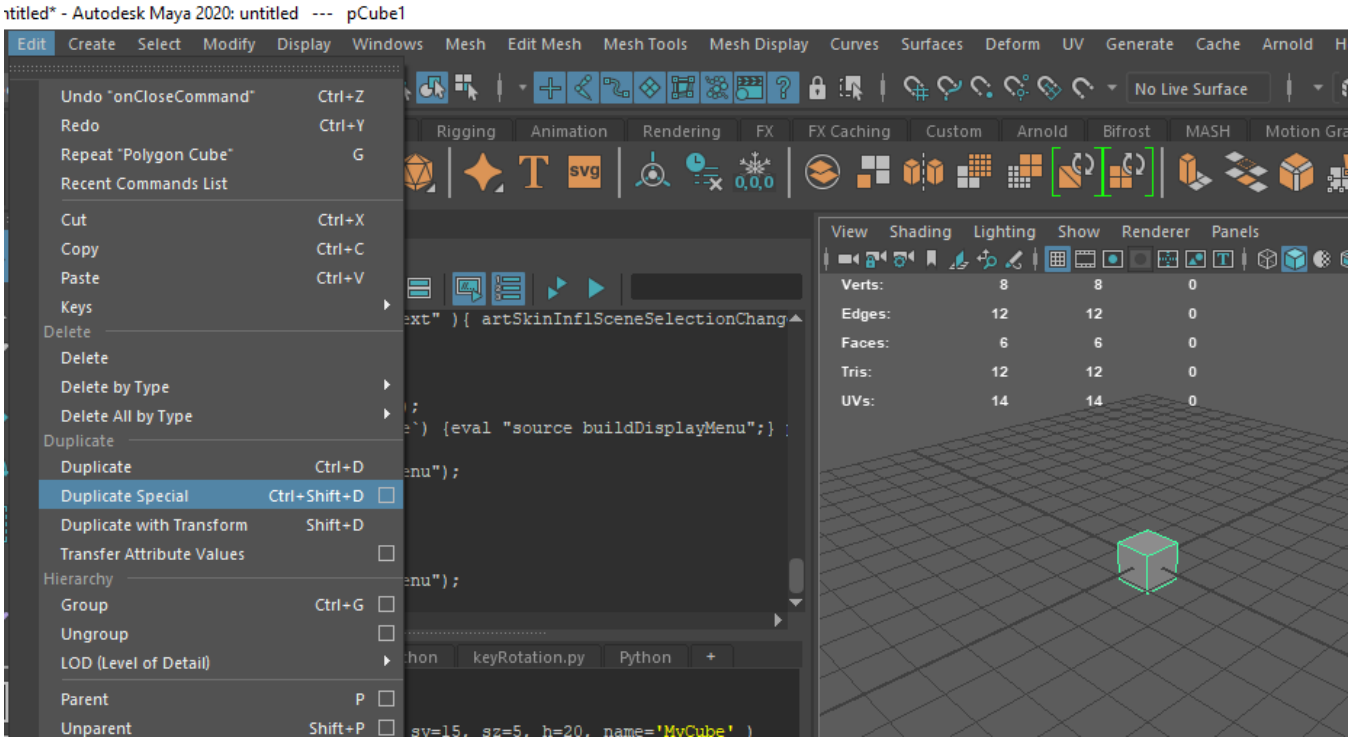
Вузол форми визначає властивості дисплея та тесселяції його вхідного об'єкта "**polyCube**".

Об'єкт '**polyCube**' містить фактичні структури даних полігону, Включаючи її вершини, ребра, площини та UV’s

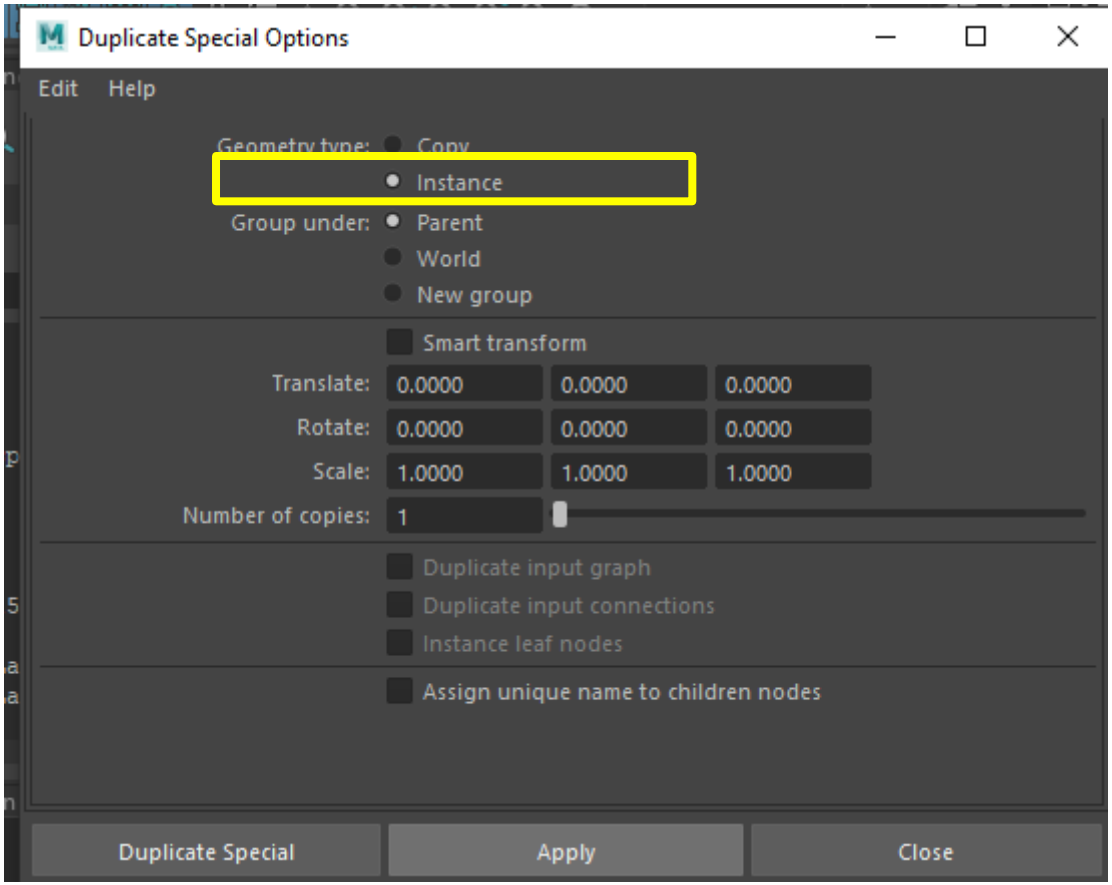
Тепер, коли ми маємо більш повне уявлення про результат команди "polyCube", ми можемо подивитися, як створити кількаекземплярів нашого куба.

Щоб зробити це вручну, виберіть куб і виберіть

Edit-Duplicate Special Option Box

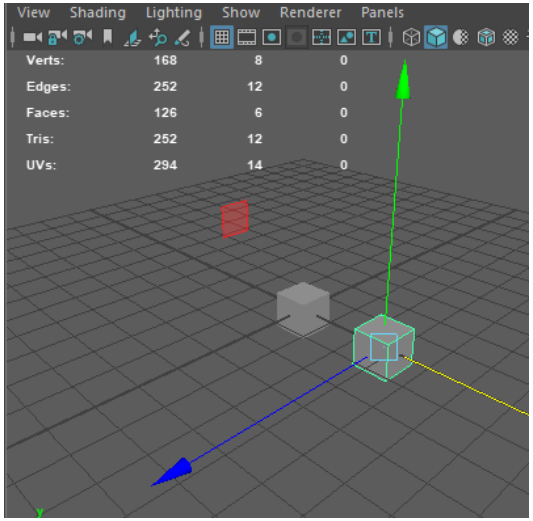


Встановіть тип геометрії в “instance" і натисніть “Apply".

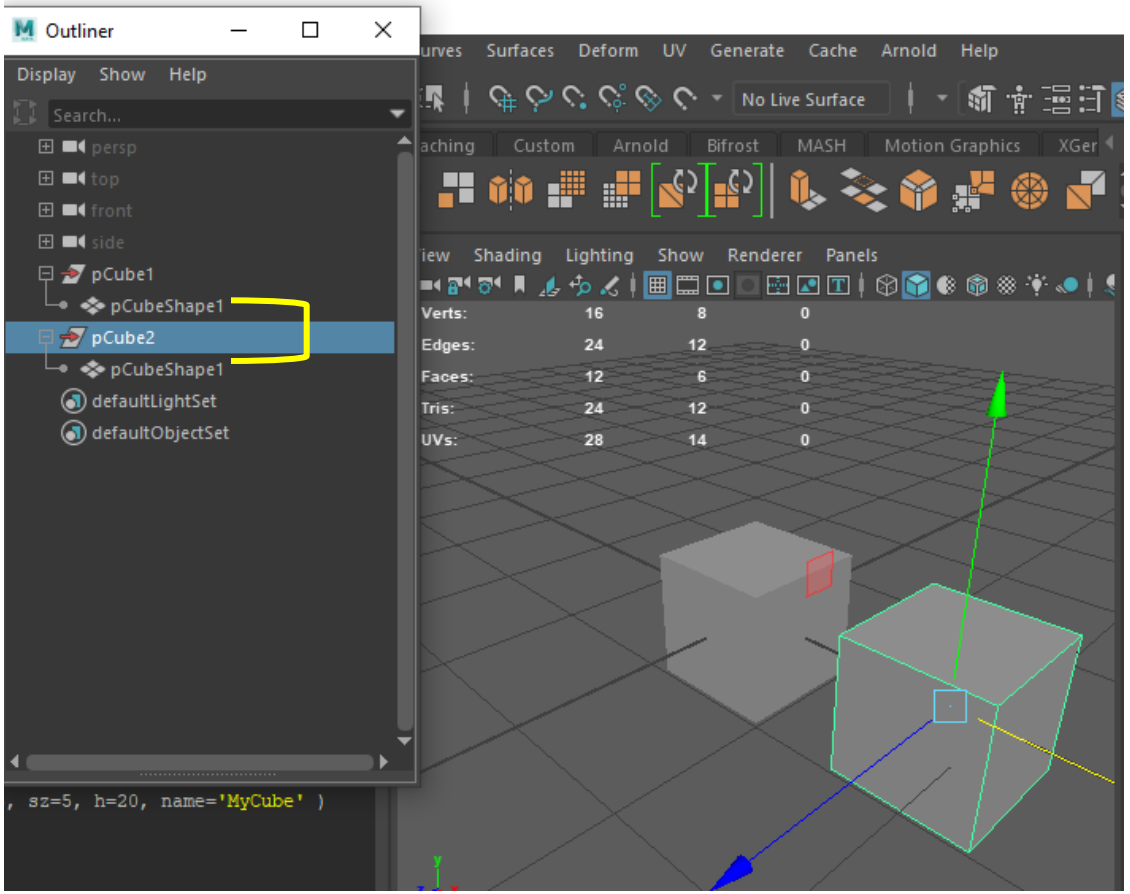


Новий куб створюєтьсянасцені в тому ж положенні, що й перший.

Давайте перенесем новий куб з першого куба.

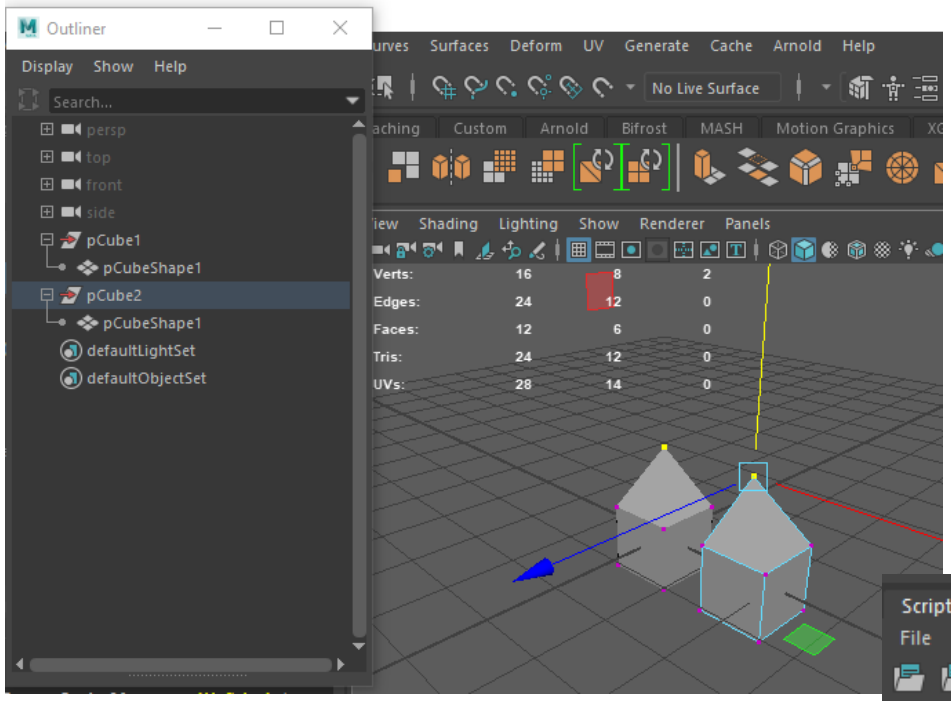


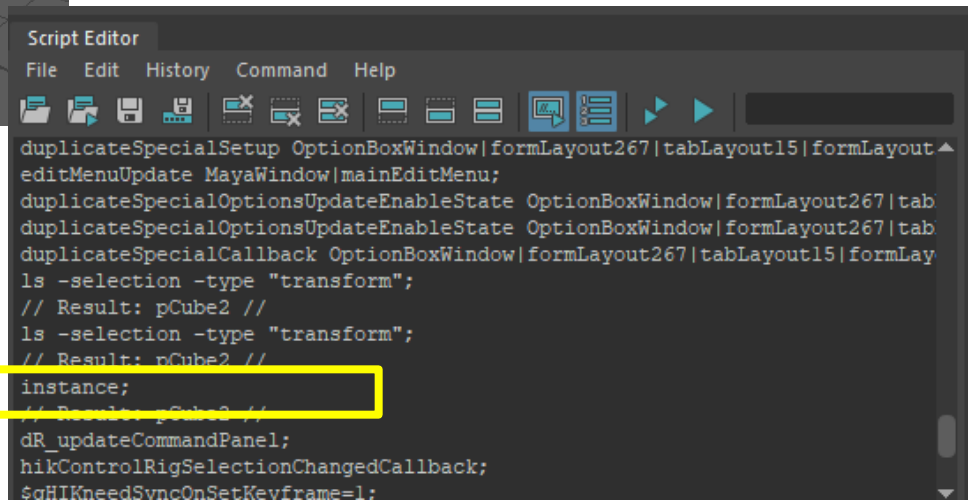
Розгорніть обидва перетворення вузлів у **Outliner**.



Звернітьувагу, щовузол перетворення ‘**polyCube2**‘ має такий самий вузол форми, що і першийкуб.

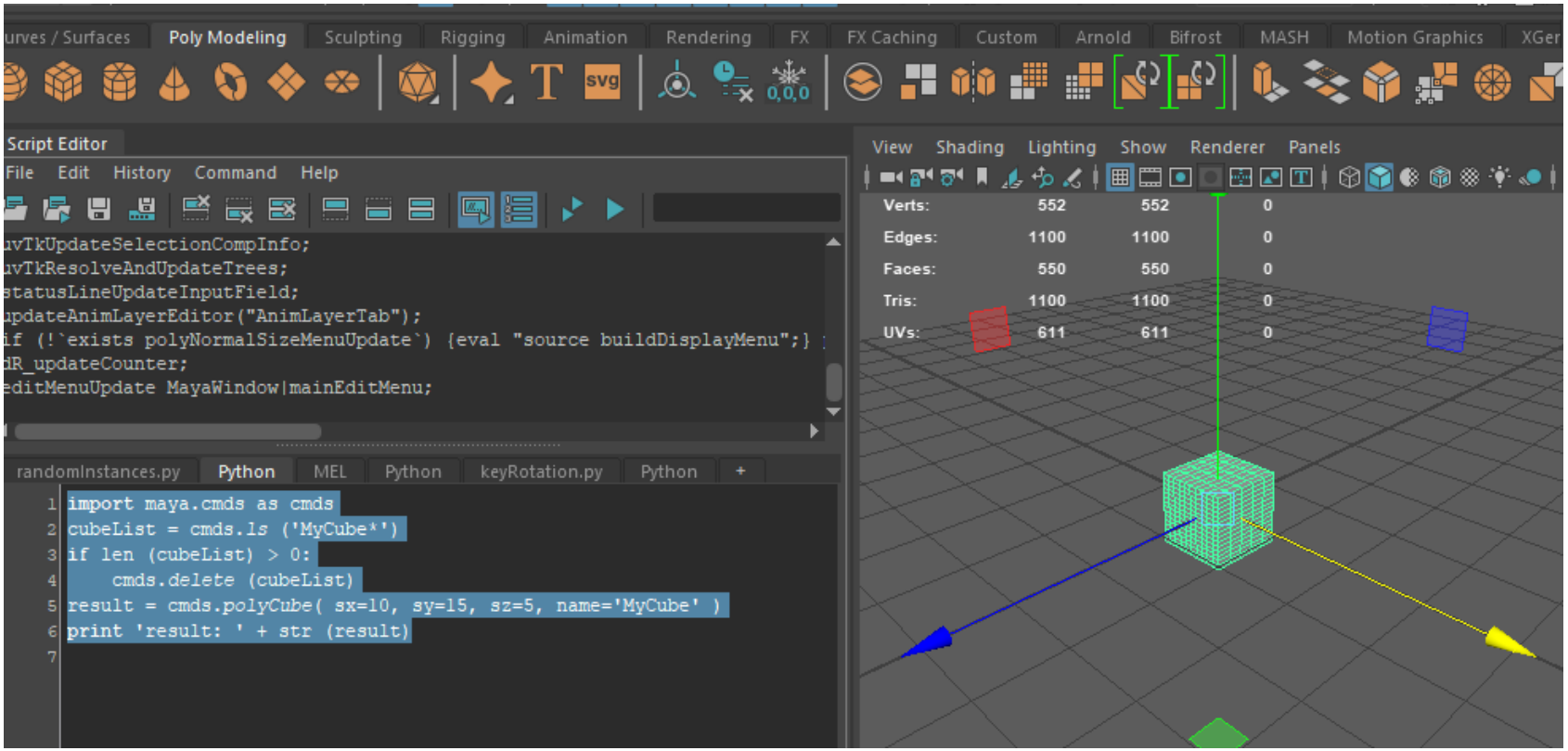
Таким чином, зміна геометрії вплине на всі випадки.





У виводі MEL зверніть увагу, що команда 'instance' була використана.

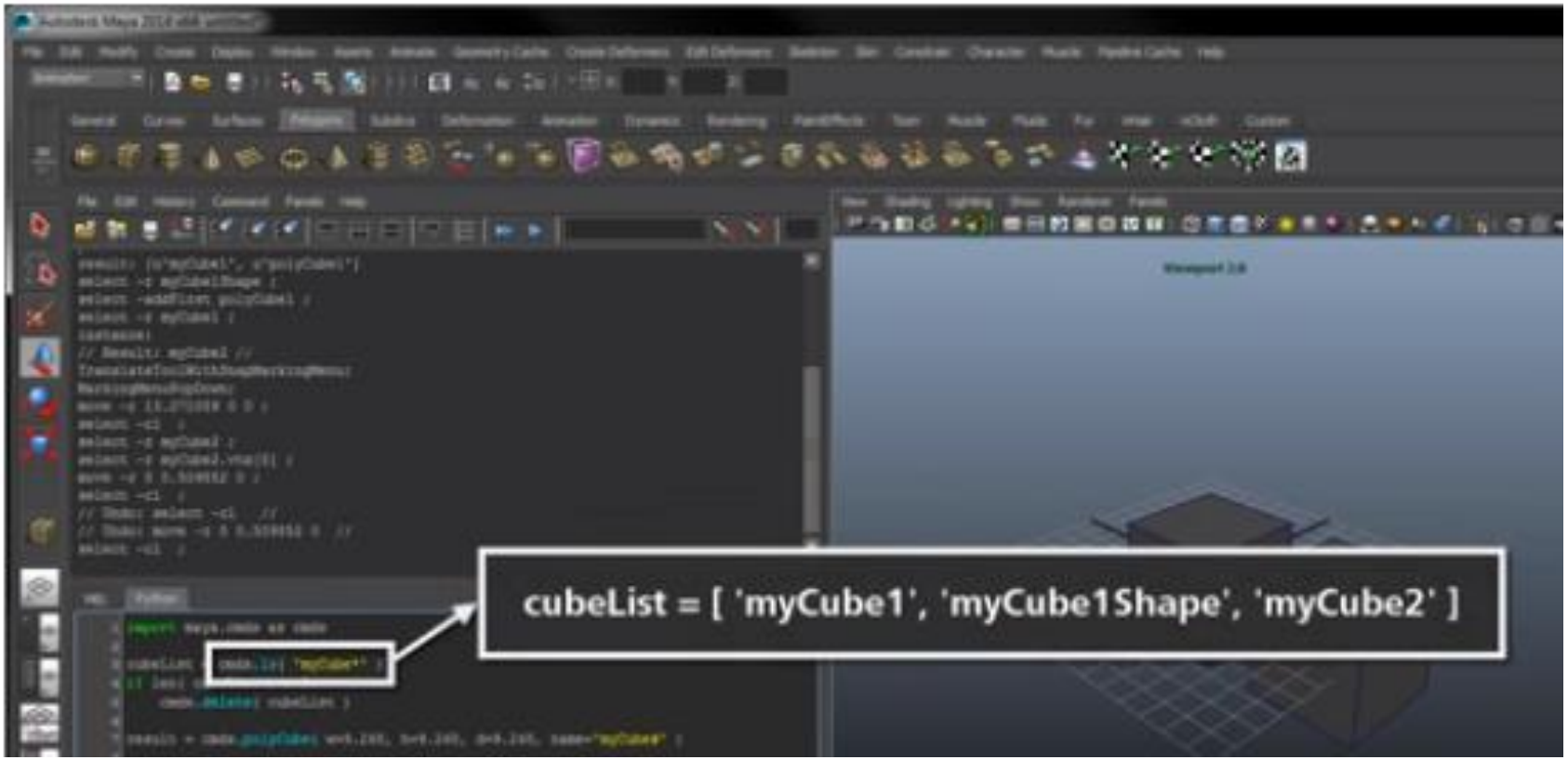
Давайте подивимося, як додати цю команду до нашого сценарію Python.



Це спрощує тестування, оскільки ми не будемо вручну видаляти всі куби, перш ніж запускати наш скрипт щоразу.

"**ls**" повертає ім'я об'єкта, яке відповідає певній вимозі.
У цьому випадку ми хочемо отримати всі об'єкти, що починаються з “MyCube".

Зірка ('*') – це символ підстановки, який відповідає будь-якому тексту після Префіксу MyCube, наприклад, 'myCube1' і такдалі.

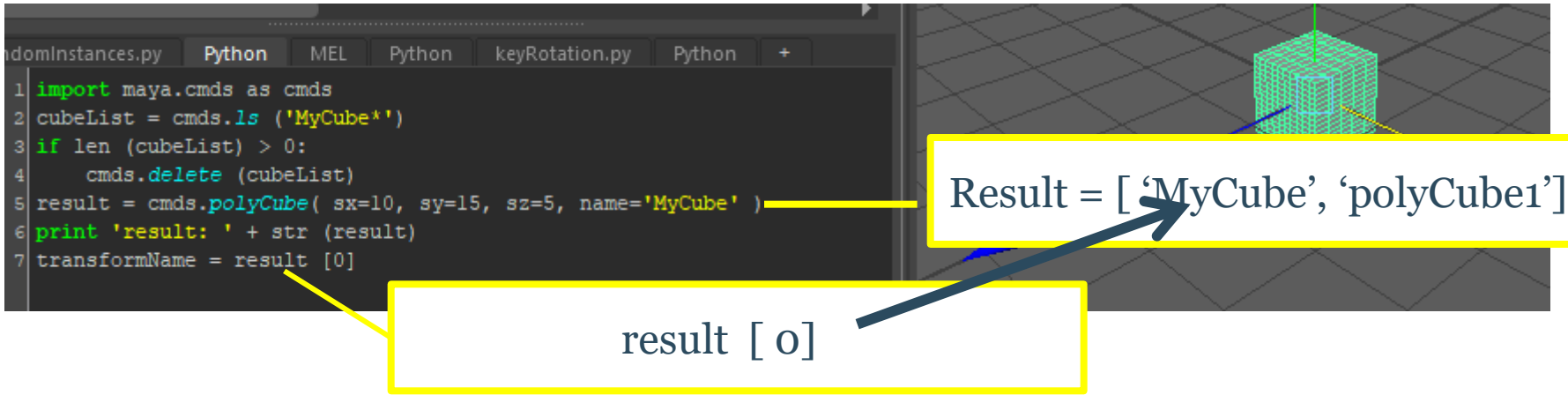


Команда "**delete**" потім видаляє кожен елемент у cubeList зі сцени.

```
import maya.cmds as cmds
cubeList = cmds.ls ('MyCube*')

if len (cubeList) > 0:
    cmds.delete (cubeList)

result = cmds.polyCube( sx=10, sy=15, sz=5, name='MyCube' )

print 'result: ' + str (result)
```

Команда “instance" приймає назву перетвореного вузла як введення.

Встановіть змінну **transformName** на перший елемент, який міститься в списку результатів, іншими словами, **myCube1**.



```
import maya.cmds as cmds
cubeList = cmds.ls ('MyCube*')

if len (cubeList) > 0:
    cmds.delete (cubeList)

result = cmds.polyCube( sx=10, sy=15, sz=5, name='MyCube' )

print 'result: ' + str (result)

transformName = result [0]

instanceResult = cmds.instance ( transformName, name=transformName + '_instance*')

print 'instanceResult:' + str (instanceResult)
```

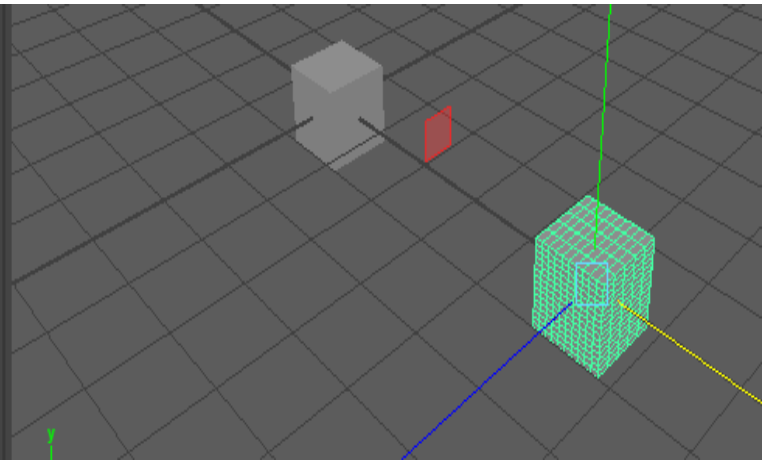
Оригінальний куб та його копія були створені в однієї позиції.

Наступним кроком перемістить новий куб на 5 одиниць по вісі x.

cmds.move (5, 0, 0, instanceResult)

domInstances.py | Python | MEL | Python | keyRotation.py | Python | +

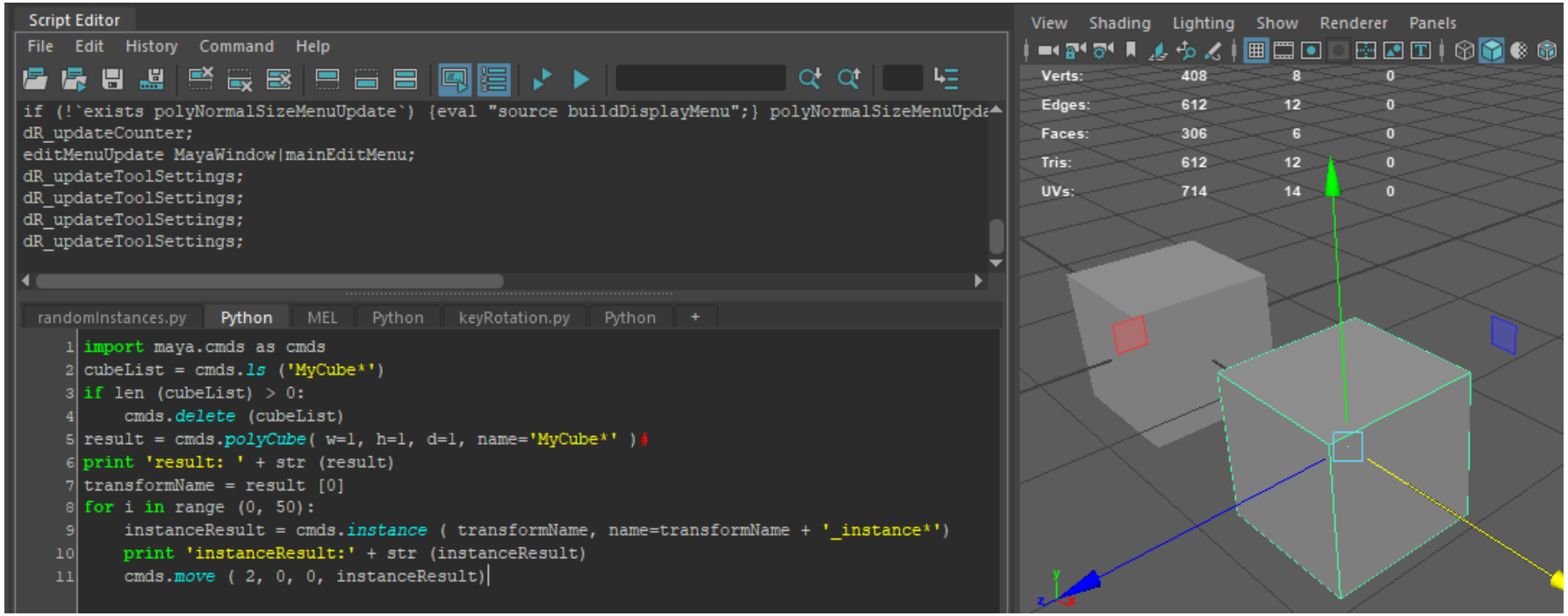
```
1 import maya.cmds as cmds
2 cubeList = cmds.ls ('MyCube*')
3 if len (cubeList) > 0:
4     cmds.delete (cubeList)
5 result = cmds.polyCube( sx=10, sy=15, sz=5, name='MyCube' )
6 print 'result: ' + str (result)
7 transformName = result [0]
8 instanceResult = cmds.instance ( transformName, name=transformName + '_instance*')
9 print 'instanceResult:' + str (instanceResult)
10 cmds.move ( 5, 0, 0, instanceResult)
```



Далі ми можемо прописувати ці команди для кожного нового куба, але що робити, коли необхідно створити п’ятдесят кубів, з новою позицією, обертанням та масштабуванням? Тоді ми можемо прописати цикл. Наприклад такий:

```
import maya.cmds as cmds
cubeList = cmds.ls ('MyCube*')

if len (cubeList) > 0:
   cmds.delete (cubeList)

result = cmds.polyCube( w=1, h=1, d=1, name='MyCube*' )

print 'result: ' + str (result)

transformName = result [0]

for i in range (0, 50):

   instanceResult = cmds.instance ( transformName, name=transformName + '_instance*')
   print 'instanceResult:' + str (instanceResult)
   cmds.move ( 10, 0, 0, instanceResult)
```

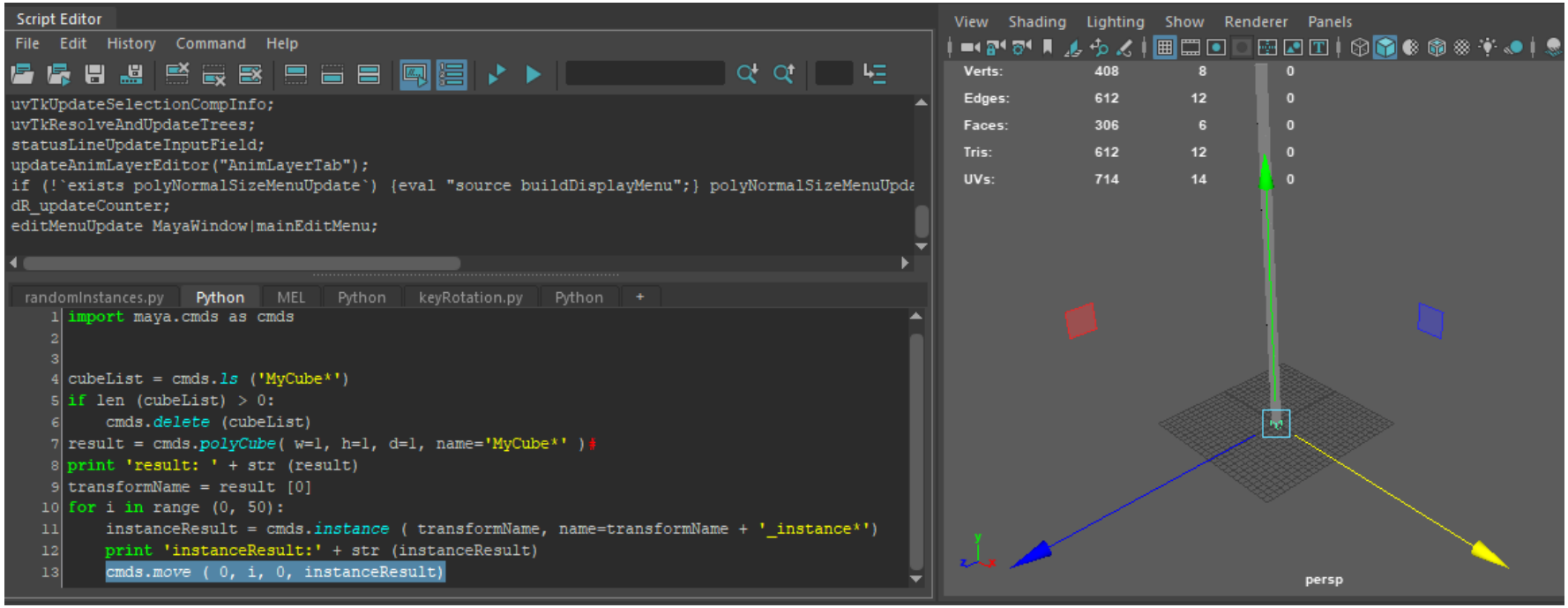
Цей діапазон призводить до того, що for-loop повторить п'ятдесят разів і збільшує значення "i" від 0 до 49



Поки ми знаходимося на цьому етапі, давайте коментувати наші дії щодо друку, оскільки ми не будемо друкувати кожен результат кожного разу, коли буде створено об'єкт. Використовуйте символ фунта, щоб коментувати один рядок.

Ми також змінюємо ширину, висоту та глибину нашого вихідного куба до 1, щоб звести до мінімуму будь-які перекриття. Замість того, щоб перемістити кожен куб на 10 одиниць від оригіналу, ми використаємо значення " і " як координати у для складання кубів примірників. Давайте розглянемо рандомізування позиції кожного куба.

```
import maya.cmds as cmds


cubeList = cmds.ls ('MyCube*')

if len (cubeList) > 0:
   cmds.delete (cubeList)

result = cmds.polyCube( w=1, h=1, d=1, name='MyCube*' )#
print 'result: ' + str (result)
transformName = result [0]

for i in range (0, 50):
   instanceResult = cmds.instance ( transformName, name=transformName + '_instance*')
   print 'instanceResult:' + str (instanceResult)

   cmds.move ( 0, i, 0, instanceResult)
```


Script Editor
File Edit History Command Help
uvTkUpdateSelectionCompInfo;
uvTkResolveAndUpdateTrees;
statusLineUpdateInputField;
updateAnimLayerEditor("AnimLayerTab");
if (!`exists polyNormalSizeMenuUpdate`) {eval "source buildDisplayMenu";} polyNormalSizeMenuUpda
dR_updateCounter;
editMenuUpdate MayaWindow|mainEditMenu;
randomInstances.py
Python
MEL
Python
keyRotation.py
Python
+
1 import maya.cmds as cmds
2
3
4 cubeList = cmds.ls ('MyCube*')
5 if len (cubeList) > 0:
6 cmds.delete (cubeList)
7 result = cmds.polyCube(w=1, h=1, d=1, name='MyCube*')#
8 print 'result: ' + str (result)
9 transformName = result [0]
10 for i in range (0, 50):
11 instanceResult = cmds.instance (transformName, name=transformName + '_instance*')
12 print 'instanceResult:' + str (instanceResult)
13 cmds.move (0, i, 0, instanceResult)
View Shading Lighting Show Renderer Panels
Verts: 408 8 0
Edges: 612 12 0
Faces: 306 6 0
Tris: 612 12 0
UVs: 714 14 0
persp

Давайте розглянемо рандомізування позиції кожного куба.

Імпортуємо випадковий модуль Python.

```
import maya.cmds as cmds
import random

cubeList = cmds.ls ('MyCube*')
if len (cubeList) > 0:
    cmds.delete (cubeList)
result = cmds.polyCube( w=1, h=1, d=1, name='MyCube*' )#
print 'result: ' + str (result)
transformName = result [0]
for i in range (0, 50):
    instanceResult = cmds.instance ( transformName, name=transformName + '_instance*')
    print 'instanceResult:' + str (instanceResult)
    cmds.move ( 0, i, 0, instanceResult)
```

Введіть random.seed (1234), щоб використовувати кожен раз, коли ми запускаємо наш сценарій, використовувати ту саму послідовність випадкових чисел. Ви можете змінити "1234" на будь-яке інше число, щоб отримати різний результат.

Генерація випадкових чисел (модуль random)

Функція random () в Python використовується для генерації псевдовипадкових чисел. Він генерує числа для деяких значень, званих seed значенням.

Як працює функція seed?

Функція початкового числа використовується для зберігання випадкового методу генерації одних і тих же випадкових чисел при багаторазовому виконанні коду на одній або різних машинах.

Початкове значення має важливе значення для комп'ютерної безпеки, оскільки воно псевдовипадково створює безпечний секретний ключ шифрування. Таким чином, використовуючи настроюється початкове значення, ви можете форматувати безпечний генератор псевдовипадкових чисел в потрібному вам місці.

Python random seed

Функція random.seed () в Python використовується для ініціалізації випадкових чисел. За замовчуванням генератор випадкових чисел використовує поточний системний час. Якщо ви двічі використовуєте один і той же початкове значення, ви отримаєте один і той же результат, що означає випадкове число двічі.

Синтаксис

```
random.seed(svalue, version)
```

Параметри

Параметр svalue є необов'язковим, і це початкове значення, необхідне для генерації випадкового числа. Значення за замовчуванням - None, і якщо None, генератор використовує поточний системний час.

Приклад

```
import random

random.seed(10)
print(random.random())

random.seed(10)
print(random.random())
```

Результат:

```
0.5714025946899135
0.5714025946899135
```

Цей приклад демонструє, що якщо ви двічі використовуєте один і той же початкове значення, ви двічі отримаєте один і той же випадкове число.
Давайте подивимося на інший приклад, в якому ми генеруємо один і той же випадкове число багато разів.

Давайте подивимося на інший приклад, в якому ми генеруємо один і той же випадкове число багато разів.

```
import random

for i in range(5):

    # Any number can be used in place of '11'.
    random.seed(11)

    # Generated random number will be between 1 to 1000.
    print(random.randint(1, 1000))
```

Результат:

```
464
464
464
464
464
```

Коли ми передаємо певну початкове число в генератор випадкових чисел, кожен раз, коли ви виконуєте програму, ви отримуєте одні й ті ж числа. Це корисно, коли вам потрібен передбачуваний джерело випадкових чисел.

Це спрощує оптимізацію кодів, коли для тестування використовуються випадкові числа. Висновок коду іноді залежить від введення. Тому використання випадкових чисел для тестування алгоритмів може бути проблематичним.

Крім того, функція seed використовується для генерації одних і тих же випадкових чисел знову і знову і спрощує процес тестування алгоритму.

random.uniform

random.uniform (<Початок "," Кінець>) - повертає псевдовипадкове дійсне число в діапазоні від <Початок> до <Кінець>:

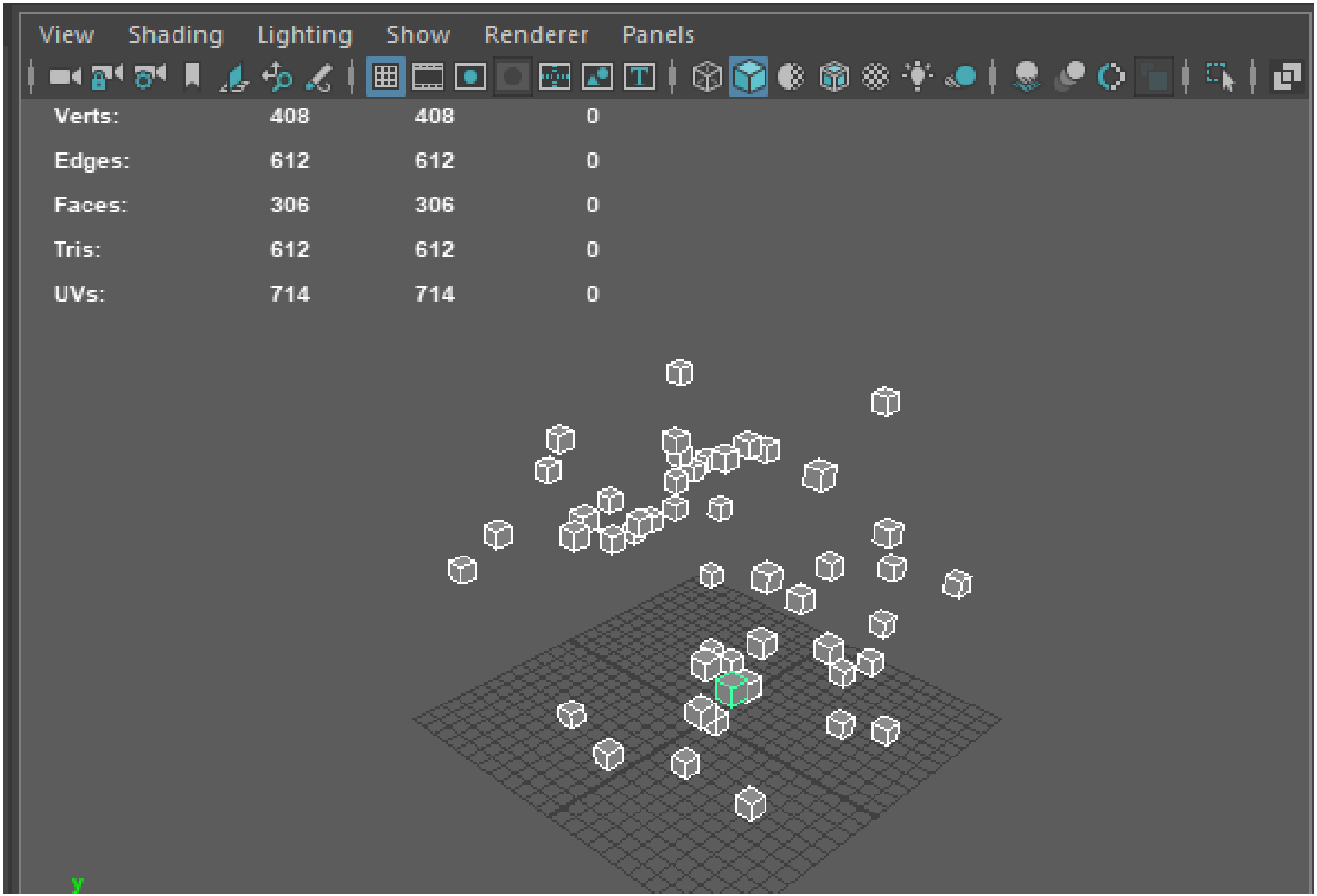
```
random.uniform(0, 20)
15.330185127252884

random.uniform(0, 20)
18.092324756265473
```

```
import maya.cmds as cmds
import random
random.seed (1234)
cubeList = cmds.ls ('MyCube*')
if len (cubeList) > 0:
    cmds.delete (cubeList)
result = cmds.polyCube( w=1, h=1, d=1, name='MyCube*' )#
print 'result: ' + str (result)
transformName = result [0]
for i in range (0, 50):
    instanceResult = cmds.instance ( transformName, name=transformName + '_instance*')
    print 'instanceResult:' + str (instanceResult)
    cmds.move ( 0, i, 0, instanceResult)
```

Значення -10 та 10 для координати x ...
Значення від 0 до 20 для координати y ...
Значення від -10 до 10 для координати z.

```
import maya.cmds as cmds
import random
random.seed ( 1234 )
cubeList = cmds.ls ('MyCube*')
if len (cubeList) > 0:
    cmds.delete (cubeList)
result = cmds.polyCube( w=1, h=1, d=1, name='MyCube*' )#
#print 'result: ' + str (result)
transformName = result [0]
for i in range (0, 50):
    instanceResult = cmds.instance ( transformName, name=transformName +
'_instance*')
    #print 'instanceResult:' + str (instanceResult)
    x = random.uniform(-10, 10)
    y = random.uniform(0, 20)
    z = random.uniform(-10,10)
    cmds.move ( x, y, z, instanceResult)
```

Група кубиків тепер випадково розпорошена над сіткою.

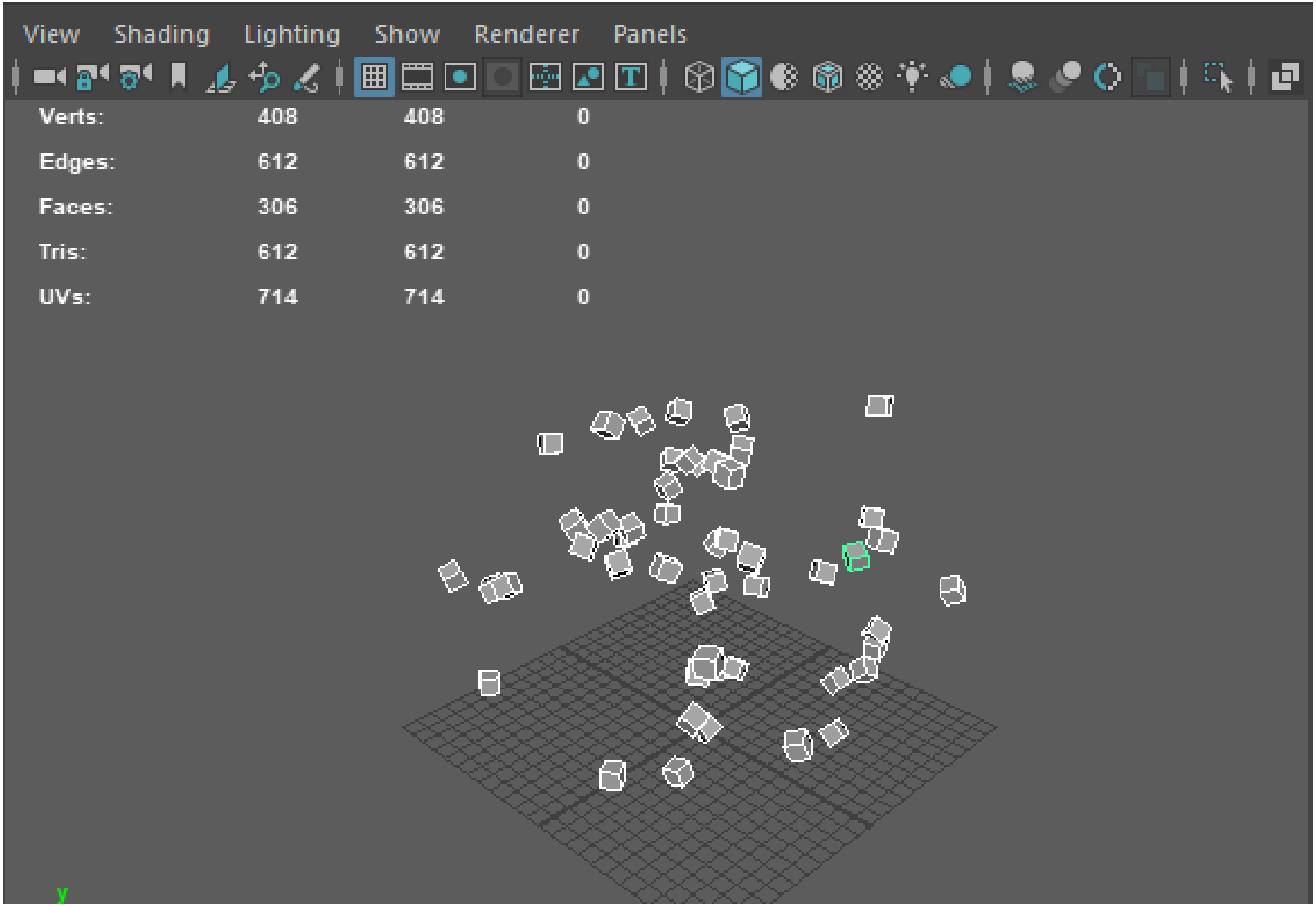


Щоб обертати кубики випадковим чином, і масштабувати їх, ми будемо використовувати ту саму техніку. Виберіть три значення від 0 до 360 для осі x, y та z обертання.

```
xRot = random.uniform (0, 360)
yRot = random.uniform (0, 360)
zRot = random.uniform (0, 360)
```

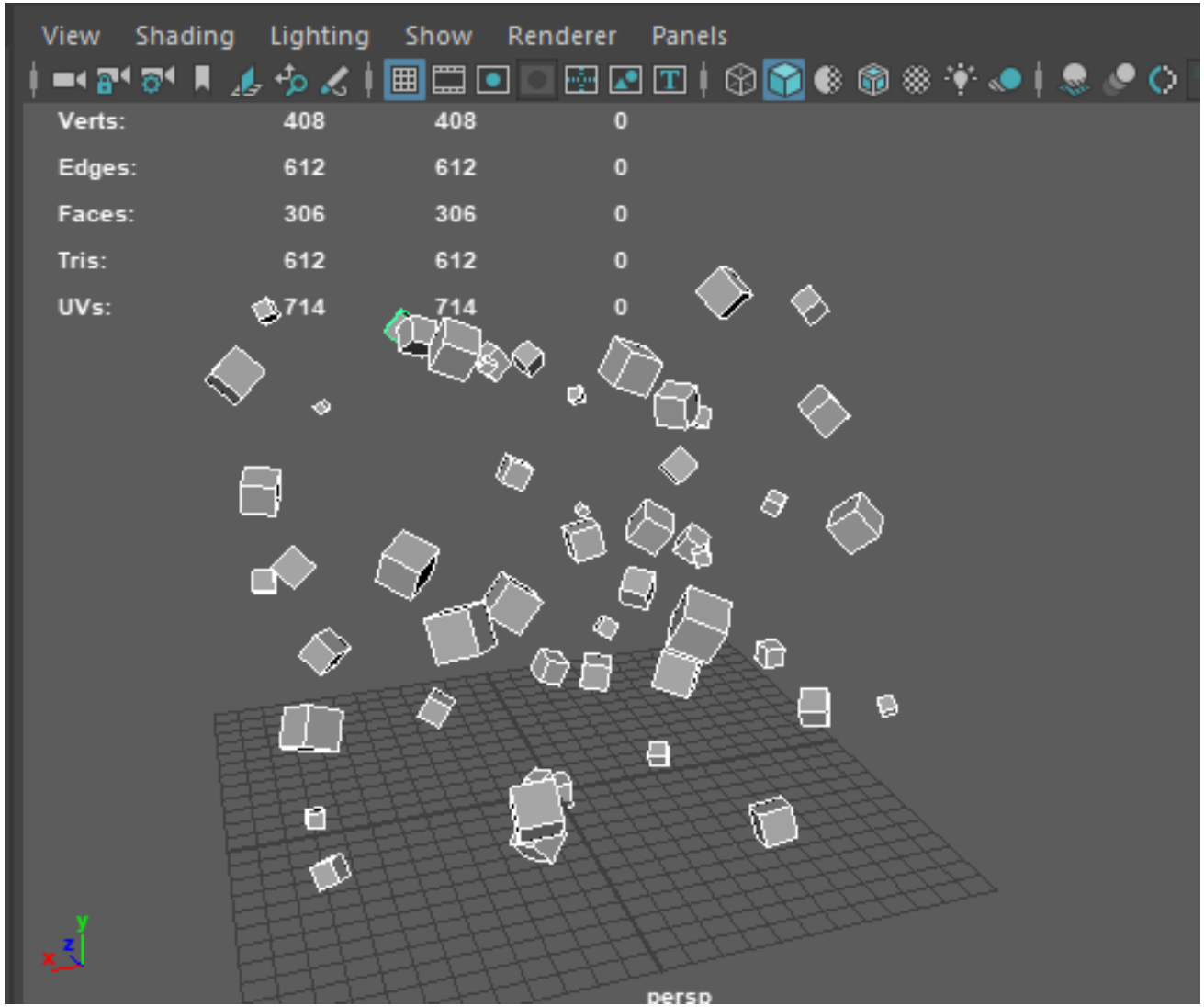
Використовуйте команду **rotate**, щоб повернути кожен екземпляр.

```
import maya.cmds as cmds
import random
random.seed ( 1234 )
cubeList = cmds.ls ('MyCube*')
if len (cubeList) > 0:
    cmds.delete (cubeList)
result = cmds.polyCube( w=1, h=1, d=1, name='MyCube*' )#
#print 'result: ' + str (result)
transformName = result [0]
for i in range (0, 50):
    instanceResult = cmds.instance ( transformName, name=transformName +
'_instance*')
    #print 'instanceResult:' + str (instanceResult)
    x = random.uniform(-10, 10)
    y = random.uniform(0, 20)
    z = random.uniform(-10,10)
    cmds.move ( x, y, z, instanceResult)
    xRot = random.uniform (0, 360)
    yRot = random.uniform (0, 360)
    zRot = random.uniform (0, 360)
    cmds.rotate (xRot, yRot, zRot, instanceResult)
```

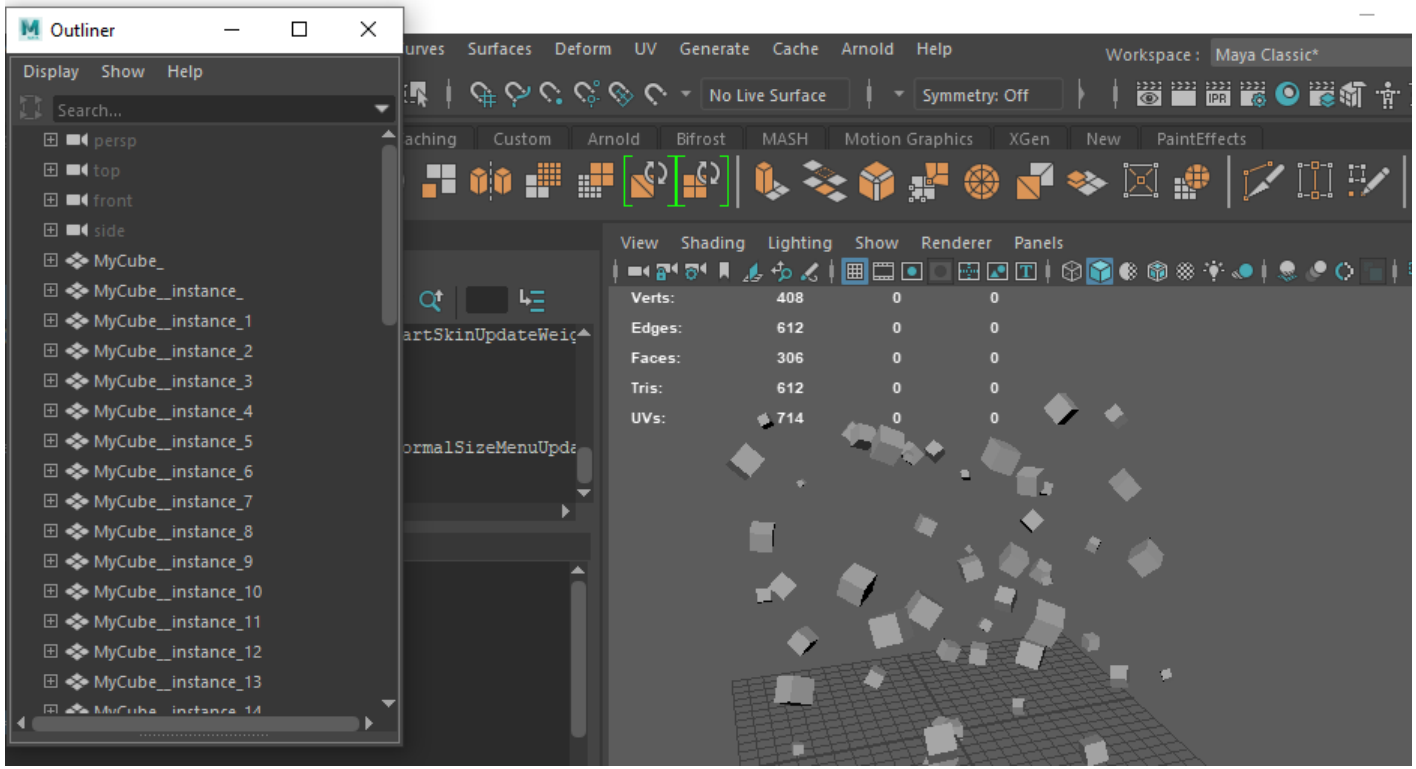

View
Shading
Lighting
Show
Renderer
Panels
Verts: 408 408 0
Edges: 612 612 0
Faces: 306 306 0
Tris: 612 612 0
UVs: 714 714 0
y

Виберіть одне значення від 0,3 до 1,5, щоб рандомізувати масштабний коефіцієнт.

```
import maya.cmds as cmds
import random
random.seed ( 1234 )
cubeList = cmds.ls ('MyCube*')
if len (cubeList) > 0:
    cmds.delete (cubeList)
result = cmds.polyCube( w=1, h=1, d=1, name='MyCube*' )#
#print 'result: ' + str (result)
transformName = result [0]
for i in range (0, 50):
    instanceResult = cmds.instance ( transformName, name=transformName + '_instance*')
    #print 'instanceResult:' + str (instanceResult)
    x = random.uniform(-10, 10)
    y = random.uniform(0, 20)
    z = random.uniform(-10,10)
    cmds.move ( x, y, z, instanceResult)
    xRot = random.uniform (0, 360)
    yRot = random.uniform (0, 360)
    zRot = random.uniform (0, 360)
    cmds.rotate (xRot, yRot, zRot, instanceResult)
    scalingFactor = random.uniform (0.3, 1.5)
   cmds.scale (scalingFactor, scalingFactor, scalingFactor, instanceResult)
```


View
Shading
Lighting
Show
Renderer
Panels
Verts: 408 408 0
Edges: 612 612 0
Faces: 306 306 0
Tris: 612 612 0
UVs: 714 714 0
persp

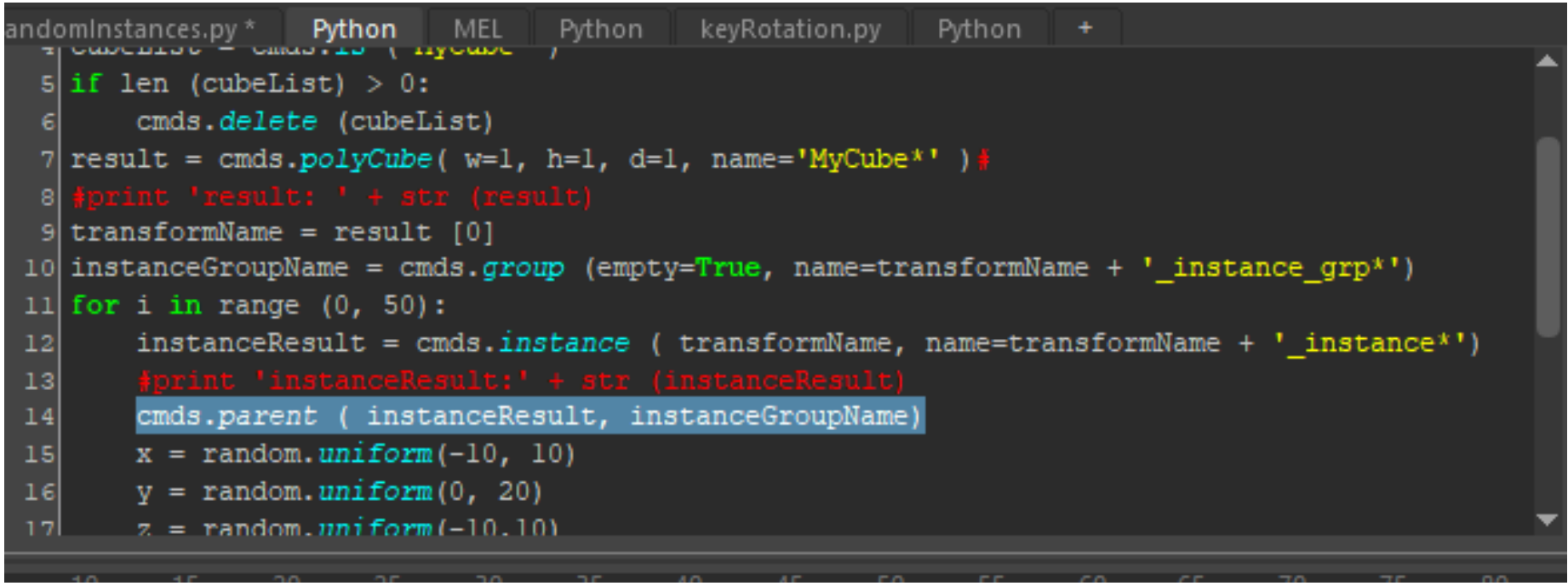
Якщо ми зараз відкриємо outliner, ми помітимо, що кубики розміщені окремо.



Останній крок - об’єднати їх разом. Використовуйте команду groupe для створення порожньої групи над циклом for. Ім’я "name" використовує назву трансформації нашого оригінального куба.

```
import maya.cmds as cmds
import random
random.seed ( 1234 )
cubeList = cmds.ls ('MyCube*')
if len (cubeList) > 0:
    cmds.delete (cubeList)
result = cmds.polyCube( w=1, h=1, d=1, name='MyCube*' )#
#print 'result: ' + str (result)
transformName = result [0]
instanceGroupName = cmds.group (empty=True, name=transformName + '_instance_grp*')
for i in range (0, 50):
```

Застововуйте parent команду, щоб додати кожен екземпляр до групи.

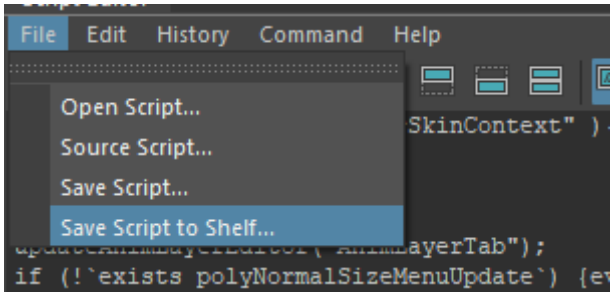


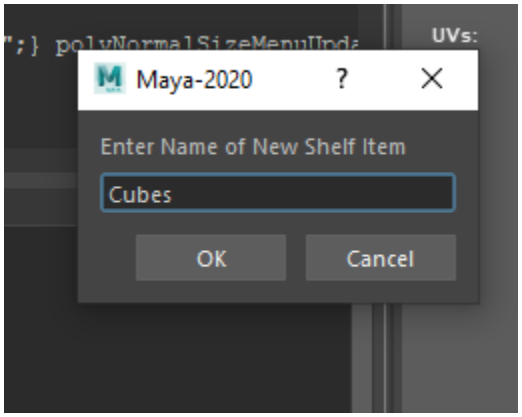
Нижче за цикл, зверніться до команди hide на оригінальний куб, щоб зробити його невидимим на сцені

randomInstances.py * | Python | MEL | Python | keyRotation.py | Python | +

```
13      #print 'instanceResult:' + str (instanceResult)
14      x = random.uniform(-10, 10)
15      y = random.uniform(0, 20)
16      z = random.uniform(-10,10)
17      cmds.move ( x, y, z, instanceResult)
18      xRot = random.uniform (0, 360)
19      yRot = random.uniform (0, 360)
20      zRot = random.uniform (0, 360)
21      cmds.rotate (xRot, yRot, zRot, instanceResult)
22      scalingFactor = random.uniform (0.3, 1.5)
23      cmds.scale (scalingFactor, scalingFactor, scalingFactor, instanceResult)
24  cmds.hide (transformName)
25
```

Давайте назвемо цей скрипт "**randomCubes.py**"
Збережіть його:





В якості останнього кроку використовуйте команду xform у instance групі, при цьому відмітку center Pivots встановлюється на True, щоб сконцентрувати його pivot.

randomInstances.py * | Python | MEL | Python | keyRotation.py | Python | +

```
13      #print 'instanceResult:' + str (instanceResult)
14      x = random.uniform(-10, 10)
15      y = random.uniform(0, 20)
16      z = random.uniform(-10,10)
17      cmds.move ( x, y, z, instanceResult)
18      xRot = random.uniform (0, 360)
19      yRot = random.uniform (0, 360)
20      zRot = random.uniform (0, 360)
21      cmds.rotate (xRot, yRot, zRot, instanceResult)
22      scalingFactor = random.uniform (0.3, 1.5)
23      cmds.scale (scalingFactor, scalingFactor, scalingFactor, instanceResult)
24  cmds.hide (transformName)
25  cmds.xform ( instanceGroupName, centerPivots=True
```