1. ВВЕДЕННЯ В МОВУ ПРОГРАМУВАННЯ PYTHON

1.1. Що таке Python?

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня з строгою динамічною типізацією. 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 компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

В стандартний пакет Python входить велика стандартна бібліотека для вирішення широкого кола завдань. В Інтернеті доступні якісні бібліотеки для Python по різних предметних областях: засоби обробки текстів і технології Інтернет, обробка зображень, інструменти для створення додатків, механізми доступу до баз даних, пакети для наукових обчислень, бібліотеки побудови графічного інтерфейсу і т.д.

1.2. Як описати мову?

Найбільш повний опис мови Python можна знайти за посиланням https://www.python.org/ .

В даному методичному посібнику пропонується розглянути мову одночасно в декількох аспектах, що досягається набором прикладів, які дозволять швидше долучитися до реального програмування, ніж в разі звичайного академічного підходу.

Однак варто звернути увагу на правильний підхід до опису мови. Створення програми - це завжди комунікація, в якій програміст передає комп'ютеру інформацію, необхідну для виконання останнім дій. Те, як ці дії розуміє програміст (тобто "сенс"), можна назвати *семантикою*. Засобом передачі цього сенсу є *синтаксис* мови програмування. Ну а те, що робить інтерпретатор на підставі переданого, зазвичай називають *прагматикою*. При написанні програми дуже важливо, щоб в цьому ланцюжку не виникало збоїв.

Синтаксис – повністю формалізована частина: його можна описати на формальній мові синтаксичних діаграм. Виявом прагматики є сам інтерпретатор мови. Саме він читає записаний відповідно до синтаксису код" і перетворює його в дію відповідно до закладеномого в ньому алгоритму. Неформальним компонентом залишається тільки семантика. Синтаксис мови Python має потужні засоби, які допомагають наблизити розуміння проблеми програмістом до її "розуміння" інтерпретатором.

1.3. Історія мови python.

Python – порівняно «молода» мова. Створюючи її у 1990-1991 роках, її автор Гвідо ван Россум (Guido van Rossum) врахував усі переваги та недоліки попередніх мов програмування. Мова почала вільно поширюватися через Інтернет і сподобалася іншим програмістам. З 1991 року Python є цілком об'єктно-орієнтованим. Python також запозичив багато рис таких мов, як C, C++ й окремі риси функціонального програмування. 3 грудня 2008 року, після тривалого тестування, вийшла перша версія Python 3 (або Python 3.0, також використовується скорочена Py3k). У Python 3 усунено багато недоліків архітектури з максимально можливим (але не повним) збереженням сумісності зі старішими версіями. На сьогодні підтримуються обидві гілки розвитку (Python 3.6 і 2.7).

У наш час Python має широку область застосування. Це – мова, яка розвивається, її використовують в реальних проектах. Засоби для роботи з Python відносяться до категорії вільно поширюваного програмного забезпечення, що гарантує відсутність претензій щодо використання «інтелектуальної власності». Існує множина засобів, які полегшують процес створення програм на Python, серед них можна виділити спеціалізовані лексичні аналізатори і редактори для програмістів (наприклад, Кате і Bluefish), інтегровані середовища розробки. Багато засобів для роботи з Пітон є кросплатформними, в конструкціях мови підтримується багатобайтове кодування (Unicode), тому програми на Python легко переносити з одного середовища функціонування на інше.

1.4. Основні алгоритмічні конструкції.

Передбачається, що слухачі вже вміють програмувати хоча б на рівнішкільної програми, і тому цілком достатньо провести паралелі міжалгоритмічними конструкціями і синтаксисом Python.

1.4.1. Послідовність операторів

Послідовні дії описуються послідовними рядками програми. Варто, щоправда, додати, що в програмах важливі відступи, тому всі оператори, що входять до послідовність дій, повинні мати один і той самий відступ:

```
>>> a = 1
>>> b = 2
>>> a = a + b
>>> b = a - b
>>> a = a - b
>>> print(a, b)
2 1
```

При роботі з Python в інтерактивному режимі як би вводиться одна велика програма, що складається з послідовних дій. В наведеному вище прикладі використані оператори присвоювання і оператор *print*.

1.4.2. Умовні оператори

```
if a > b:
   c = a
else:
      c = b
```

Цей шматок коду на Python інтуїтивно зрозумілий кожному, хто пам'ятає, що *if* по англійськи означає "якщо", а *else* - "інакше". Оператор розгалуження має в даному випадку дві частини, оператори кожної з яких записуються з відступом вправо щодо оператора розгалуження. Більш загальний випадок - оператор вибору - можна записати за допомогою наступного синтаксису:

```
if a < 0:
   s = -1
elif a == 0:
   s = 0
else:
   s = 1
```

Варто зауважити, що elif - це скорочений else if. Без скорочення довелося б застосовувати вкладений оператор розгалуження:

```
if a < 0:
   s = -1
else:
   if a == 0:
      s = 0
   else:
      s = 1
```

На відміну від оператора print, оператор if-else - складений оператор.

1.4.3. Цикли

Третьою необхідною алгоритмічною конструкцією є цикл. За допомогою циклу можна описати повторювані дії. В Python є два види циклів: цикл ПОКИ (виконується деяка умова) і цикл ДЛЯ (всіх значень послідовності). Наступний приклад ілюструє цикл ПОКИ на Python:

```
s = "abcdefghijklmnop"
```

```
while s != "":
   print(s)
   s = s[1:-1]
```

Оператор while говорить інтерпретатору Python: "поки умова циклу вірна, виконуй тіло циклу". У мові Python тіло циклу виділяється відступом. Кожне виконання тіла циклу називається ітерацією. У наведеному прикладі забирається перший і останній символ рядка до тих пір, поки не залишиться порожній рядок.

Для більшої гнучкості при організації циклів застосовуються оператори *break* (перервати) і *continue* (продовжити). Перший скасовує виконання циклу, а другий - продовжує цикл, перейшовши до наступної ітерації (якщо, звичайно, виконується умова циклу).

Наступний приклад читає рядки з файлу і виводить ті, у яких довжина більше 5 символів:

```
f = open("file.txt", "r")
while true:
   l = f.readline()
   if not l:
      break
   if len(l) > 5:
      print(l)
f.close()
```

У цьому прикладі нескінченний цикл переривається тільки при отриманні з файлу порожнього рядка (l <=5), що позначає кінець файлу.

У мові Python логічне значення несе кожен об'єкт: нулі, порожні рядки і послідовності, спеціальний об'єкт None і логічний літерал False мають значення "неправда", а інші об'єкти значення "істина". Для позначення істини зазвичай використовується 1 або True.

Цикл ДЛЯ виконує тіло циклу для кожного елемента послідовності. В наступному прикладі виводиться таблиця множення:

```
for i in range(1, 10):
   for j in range(1, 10):
      print("%2i" % (i*j))
   print()
```

Тут цикли *for* є вкладеними. Функція *range()* породжує список цілих чисел з

напіввідкритого інтервалу [1, 10). Перед кожною ітерацієюлічильник циклу отримує чергове значення з цього списку. Напіввідкриті діапазони загальноприйняті в Python. Вважається, що їх використання більш зручно і викликає менше програмістів помилок. Наприклад, *range(len(s))* породжує список індексів для списку s (в Python-послідовності перший елемент має індекс 0). Для красивого виведення таблиці множення застосована операція форматування *%* (для цілих чисел той же символ використовується для позначення операції взяття залишку від ділення).

1.4.4. Функції

Програміст може визначати власні функції двома способами: за допомогою оператора *def* або прямо в виразі, за допомогою *lambda*. Наведемо приклад визначення і виклику функції:

```
def price(hrn, kop=0):
    return "%i hrn. %i kop." % (hrn, kop)

print(price(8, 50))
print(price(7))
print(price(hrn=23, kop=70))
```

У цьому прикладі визначена функція двох аргументів (у тому числі другий має значення за замовчуванням - 0). Варіантів виклику цієї функції з конкретними параметрами також кілька. Варто лише зауважити, що при виконанні функції спочатку повинні йти позиційні параметри, а потім, іменовані. Аргументи зі значеннями за замовчуванням повинні слідувати після звичайних аргументів. Оператор ***return*** повертає значення функції. З функції можна повернути тільки один об'єкт, але він може бути кортежем з кількох об'єктів.

1.4.5. Винятки

У сучасних програмах передача управління відбувається не завжди так гладко, як в описаних вище конструкціях. Для обробки особливих ситуацій (таких як ділення на нуль або спроба зчитати неіснуючий файл) застосовується механізм винятків. Найкраще пояснити синтаксис оператора *try-except* можна наступним прикладом:

```
try:
    res = int(open('a.txt').read()) / int(open('c.txt').read())
    print(res)
except IOError:
    print("Input/outpet error")
except ZeroDivisionError:
    print("Zero division Error")
```

```
except KeyboardInterrupt:
   print("Keyboard Interrupt")
except:
      print("Error")
```

У цьому прикладі числа зчитуються з двох файлів і діляться один на одне. Внаслідок цих дій може виникнути кілька виняткових ситуацій, деякі з них відзначені в *except* (тут використані стандартні вбудовані виключення Python). Останній *except* в цьому прикладі вловлює всі інші винятки, що не були спіймані вище. Наприклад, якщо хоча б в одному з файлів знаходиться нечислове значення, функція *int()* порушить виняток *ValueError*, який зможе відловити останній *except*. Виконання частини *try*, в разі виникнення помилки, після виконання однієї з частин *except* вже не здійснюється

Винятки можна порушувати і з програми. Для цього служить оператор *raise*:

```
class MyError(Exception):
   pass
try:
   raise MyError("my error 1")
except MyError:
   print("Error:")
      raise
```

Для тверджень застосовується спеціальний оператор *assert*. він призводить до виключення типу *AssertionError*, якщо задана в ньому умова невірна. Цей оператор використовують для самоперевірки програми:

```
a = 1
b = 9
c = a + b
assert c == a + b
```

Крім описаної форми оператора, є ще форма try-finally для гарантованого виконання деяких дій при передачі управління зсередини оператора try-finally зовні. Він може застосовуватися для звільнення зайнятих ресурсів, що вимагає обов'язкового виконання, незалежно від того, що сталися всередині:

```
try:
...
finally:
   print("The programm has been done")
```

1.5. Вбудовані типи даних.

Всі дані в Python представлені об'єктами. Імена є лише посиланнями на ці об'єкти і не несуть навантаження по декларації типу. Значення вбудованих типів мають спеціальну підтримку в синтаксисі мови: можна записати літерал рядка, числа, списку, кортежу, словника. Синтаксичну ж підтримку операцій над вбудованими типами можна легко зробити доступною і для об'єктів визначених користувачами класів.

Слід також зазначити, що об'єкти можуть бути змінними (mutable) та незмінними (immutable). Наприклад, рядки в Python є незмінними, тому операції над рядками створюють нові рядки.

Карта вбудованих типів (з іменами функцій для приведення до потрібного типу і іменами класів для наслідування від цих типів):

- спеціальні типи: None, NotImplemented і Ellipsis;
- числа;
  - цілі
    - звичайне ціле int
    - ціле довільної точності long
    - логічний bool
  - число з плаваючою точкою float
  - комплексне число complex
- послідовності;
  - незмінні:
    - рядок str;
    - Unicode-рядок unicode;
    - кортеж tuple;
  - змінні:
    - список list;
- відображення:
  - словник dict
- об'єкти, які можна викликати:
  - функції (призначені для користувача і вбудовані);
  - функції-генератори;
  - методи (призначені для користувача і вбудовані);
  - класи (нові та "класичні");
  - екземпляри класів (якщо мають метод __call__);
- модулі;
- класи (див. Вище);
- екземпляри класів (див. Вище);
- файли file;
- допоміжні типи buffer, slice.

Дізнатися тип будь-якого об'єкта можна за допомогою вбудованої функції *type()*.

1.5.1. Числа

Python підтримує як цілі, так і дробові числа. Вказувати тип числа явно не потрібно, Python сам визначить його за наявністю чи відсутністю десяткової крапки.

>>> type(1)
<class 'int'>

Функцію type() можна використовувати, щоб визначати тип значення змінної. Як можна було очікувати, 1 - ціле (int).

>>> isinstance(1, int)
True

Для перевірки приналежності типу можна використовувати функцію isinstance().

>>> 1 + 1
2

Додавання двох цілих дає нам цілий результат.

>>> 1 + 1.0
2.0
>>> type(2.0)
<class 'float'>

Результатом додавання цілого та дробового числа є дробове. Для виконання додавання Python приводить ціле число до типу float і повертає результат цього ж типу.

Деякі оператори (такі, як додавання) за потреби перетворюють цілі числа в дробові. Ви можете зробити це саме самостійно.

>>> float(2)
2.0

Можна явно перетворювати int у float, використовуючи функцію float()

>>> int(2.0)
2

Дробове число можна перетворити на ціле за допомогою функції int

>>> int(2.5)
2

Функція int просто відкидає дробову частину, а не округлює.

>>> int(-2.5)
-2

Функція int для від'ємних повертає найменше ціле число, більше або рівне даному (тобто теж просто відкидає дробову частину). Тому не плутайте її з функцією math.floor.

>>> 1.12345678901234567890
1.1234567890123457

Десяткові дроби мають точність до 15 знаків після коми.

>>> type(1000000000000000)
<class 'int'>

Цілі можуть бути як завгодно великими.

Числові операції

>>> 11 / 2
5.5

Оператор / виконує ділення чисел з плаваючою крапкою. Він повертає float, навіть якщо чисельник та знаменник цілі.

>>> 11 // 2
5

Оператор // виконує цілочисельне ділення. Коли результат додатній, ви можете вважати його відкиданням (не округленням) дробової частини звичайного ділення, але будьте обережними з цим.

>>> −11 // 2
−6

При цілочисельному діленні від'ємних чисел оператор // округлює "вгору" до найближчого цілого. Хоча, говорячи формально, він округлює вниз, тому що -6 менше за -5.

>>> 11.0 // 2
5.0

Оператор // не завжди повертає цілі. Якщо чисельник чи знаменник дробові, // повертає float, яке, щоправда, все одно округлене до найближчого цілого.

>>> 11 ** 2
121

Оператор ** означає піднесення до степеня. 11^2=121.

>>> 11 % 2
1

Оператор % повертає остачу від цілочисельного ділення.

Крім арифметичних операцій, можна використовувати операції з модуля math.

1.5.2. Тип bool

Булеві змінні приймають лише істинне чи хибне значення. Python має для цих значень дві відповідні константи: True та False, які можна використовувати для присвоєння значень змінним. Окрім констант, булеві значення можуть приймати вирази. А в деяких місцях (наприклад, в операторі if) Python навіть очікує того, що значення виразу можна буде привести до булевого типу. Такі місця називаються булевими контекстами. Ви можете використати майже будь-який вираз в булевому контексті, і Python намагатиметься визначити його істинність. Різні типи даних мають різні правила щодо того, які значення є істинними, а які - хибними в булевому контексті.

```
for i in (False, True):
   for j in (False, True):
      print( i, j, ":", i and j, i or j, not i)
```

Результат:

False False : False False True
False True : False True True
True False : False True False
True True : True True False

1.5.3. Тип string та тип unicode

Python рядки бувають двох типів: звичайні і Unicode-рядка. Фактично рядок - це послідовність символів (в разі звичайних рядків можна сказати "послідовність байтів"). Рядки-константи можна задати в програмі за допомогою строкових літералів. Для литералів нарівні використовуються як апострофи ('), так і звичайні подвійні лапки ("). Для багаторядкових литералів можна використовувати потроєння апострофи або потроєння лапки. Керуючі послідовності всередині строкових

літералів задаються зворотною косою межею (\). Приклади написання строкових літералів:

```
s1 = "строка1"
s2 = 'строка2\nз переводом строки на наступний рядок'
s3 = """строка3
переводом строки на наступний рядок """
u1 = u'\u043f\u0440\u0438\u0432\u0435\u0442'
u2 = u'Ще один приклад
```

Операції над рядками включають конкатенацію "+", повтор "*", форматування "%". Також рядки мають велику кількість методів, деякі з яких наведено нижче. Повний набір методів (та їх необов'язкових аргументів) можна отримати в документації по Python.

```
>>> "A" + "B"
'AB'
>>> "%s %i" % ("abc", 12)
'abc 12'
>>> "A"*10
'AAAAAAAAAA'
```

1.5.4. Тип tuple

Для представлення константної послідовності (різнорідних) об'єктів використовується тип кортеж. Літерал кортежу зазвичай записується в круглих дужках, але можна, якщо не виникають неоднозначності, писати та без них. Приклади запису кортежів:

```
p = (1.2, 3.4, 0.9)
for s in "one", "two", "three":
   print (s)
one_item = (1,)
empty = ()
p1 = 1, 3, 9 # без дужок
p2 = 3, 8, 5, # кома в кінці кортежу ігнорується
```

Використовувати синтаксис кортежів можна і в лівій частині оператораприсвоювання. У цьому випадку на основі обчислених справа значень формується кортеж та зв'язується один в один з іменами в лівій частині.

```
a, b = b, a
```

1.5.5. Тип list

В Python немає масивів з довільним типом елемента. Замість них використовуються списки. Їх можна задати за допомогою літералів, що записуються в квадратних дужках, або за допомогою спискових включень.

lst1 = [1, 2, 3,]
lst2 = [x**2 for x in range(10) if x % 2 == 1]
lst3 = list("abcde")

1.5.6. Робота з послідовностями

Таблиця 1.1 – операції та методи для роботи з послідовностями

Операція	**Пояснення**
len(s)	Довжина послідовності s
x in s (x not in s)	Перевірка приналежності елемента послідовності. Повертає True або False
s + s1	Конкатенація послідовностей s та s1
s*n, n*s	Послідовність з n раз повтореної послідовності s. Якщо n <0, повертається порожня послідовність
s[i]	Повертає i -й елемент s або len (s) -i -й, якщо i <0
s[i:j:d]	Зріз з послідовності s від i до j з кроком d
min(s)	Найменший елемент s
max(s)	Найбільший елемент s
s[i] = x	i -й елемент списку s замінюється на x
s[i:j:d] = t	Зріз від i до j (з кроком d) замінюється на (список) t
del s[i:j:d]	Видалення елементів зрізу з послідовності

Таблиця 1.2 – операції та методи для роботи зі змінними послідовностями

Метод	**Пояснення**
append(x)	Додає елемент в кінець послідовності
count(x)	Рахує кількість елементів, рівних x

extend(s)	Додає в кінець даної послідовності послідовність s
index(x)	Повертає найменше i, при якому s [i] == x.
insert(i, x)	Вставляє елемент x в i -й елемент
pop(i)	Повертає i -й елемент, видаляючи його з послідовності
reverse(s)	Змінює порядок елементів s на зворотний
sort([cmpfunc])	Сортує елементи s за заданою функцією cmpfunc

1.5.7. Тип dict

Словник (хеш, асоціативний масив) - це змінна структура даних для зберігання пар ключ-значення, де значення однозначно визначається ключем. Ключем може виступати незмінний тип даних (число, рядок, кортеж і т.п.). Порядок пар ключ-значення довільний. Нижче наведено літерал для словника і приклад роботи зі словником:

```
d = {1: 'one', 2: 'two', 3: 'three', 4: 'four'}
d0 = {0: 'zero'}
print(d[1]) # береться значення по ключу
d[0] = 0 # присвоюється значення по ключу
del d[0] # видаляється пара ключ-значення з даними ключем
print(d)
for key& val in d|items()^ # цикл по всьому словнику
   print (key, val)
for key in d.keys(): # цикл по ключам словника
   print (key, d[key])
for val in d.values(): # цикл по значенням словника
   print (val)
d.update(d0) # доповнення словника іншим словником
print (len(d)) # виводить кількість елементів ключ–значення
```

1.5.8. Тип file

Об'єкти цього типу призначені для роботи із зовнішніми даними. В простому випадку - це файл на диску. Файлові об'єкти повинні підтримувати основні методи: read(), write(), readline(), readlines(), seek(), tell(), close() і т.д.

Наступний приклад показує копіювання файлу:

```
f1 = open("file1.txt", "r")
f2 = open("file2.txt", "w")
for line in f1.readlines():
   f2.write(line)
```

f2.close()
f1.close()

Варто зауважити, що крім власне файлів в Python використовуються і файлоподібні об'єкти. В дуже багатьох функціях просто неважливо, переданий їй об'єкт типу file або іншого типу, якщо він має всі ті ж методи. Наприклад, копіювання вмісту за посиланням (URL) в файл file2.txt можна досягти, якщо замінити перший рядок на:

import urllib
f1 = urllib.urlopen("https://python.org")

1.6. Вирази.

В сучасних мовах програмування прийнято проводити більшу частину обробки даних у виразах. Синтаксис виразів у багатьох мов програмування приблизно однаковий.

Пріоритет операцій показаний в наступній таблиці (в порядку зменшення). Для унарних операцій x позначає операнд. Асоціативність операцій в Python - зліва-направо, за винятком операції піднесення до степеня (**), яка читається справа наліво.

Таблиця 1.3 – Вирази мови програмування Python

lambda	лямбда-вираз
or	логічне АБО
and	логічне І
not x	логічне НІ
in, not in	перевірка приналежності
is, is not	перевірка ідентичності
<,<=,>,>=,!=,==	порівняння
|	побітовое АБО
^	побітовое виключаюче АБО
&	побітовое І
<<, >>	побітові зсуви
*, /, %	додавання і віднімання
+, -	множення, ділення, залишок
+x, -x	унарний плюс і зміна знака
~x	побітовое НЕ
**	піднесення до степеня

x.атрибут	посилання на атрибут
x [індекс]	взяття елемента за індексом
x [від: до]	виділення зрізу (від і до)
f (аргумент, ...)	визов функції
(...)	дужки або кортеж
[..]	список або спискове включення
{кл: зн ...}	словник пар ключ–значення
`вираз`	перетворення до рядка (repr)

Таким чином, порядок обчислень операндів визначається такими правилами:

- Операнд зліва обчислюється раніше операнда справа у всіх бінарних операціях, крім зведення в ступінь.
- Порівнянь виду ***a <b <c ... y <z*** фактично рівносильна: ***(a <b) and (b <c) and ... and (y <z).***
- Перед фактичним виконанням операції обчислюються потрібні для неї операнди. В більшості бінарних операцій попередньо обчислюються обидва операнда (спочатку лівий), але операції or і and, а також порівняння обчислюють таку кількість операндів, яка достатня для отримання результату.
- Аргументи функцій, вирази для списків, кортежів, словників і т.п. обчислюються зліва-направо, в порядку проходження в виразі.

Вирази завжди має результат, хоча в деяких випадках (коли вираз обчислюється заради побічних ефектів) цей результат може бути "порожнім" - None.

1.7. Імена.

Ім'я може починатися з літери (будь-якого регістра) або підкреслення, а далі допустимо використання цифр. В якості ідентифікаторів можна застосовувати ключові слова мови і небажано перевизначати вбудовані імена. Список ключових слів можна дізнатися таким чином:

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Імена, що починаються з підкреслення або двох підкреслень, мають особливий сенс. Одиночне підкреслення говорить програмісту про те, що ім'я має місцеве застосування, і не повинно використовуватися за межами модуля. Подвійним підкресленням на початку і в кінці зазвичай наділяються спеціальні імена атрибутів.

В кожній точці програми інтерпретатор "бачить" три простори імен: локальне, глобальне і вбудоване. Простір імен - відображення з імен в об'єкти.

Для розуміння того, як Python знаходить значення деякої змінної, необхідно ввести поняття блоку коду. В Python блоком коду є те, що виконується як єдине ціле, наприклад, тіло визначення функції, класу або модуля.

Локальні імена - імена, яким присвоєно значення в даному блоці коду. Глобальні імена - імена, які визначаються на рівні блоку коду визначення модуля або ті, які явно задані в операторі global. Вбудовані імена - імена зі спеціального словника __ builtins __.

Області видимості імен можуть бути вкладеними один в одного, наприклад, всередині викликаної функції видно імена, які визначені в визиваючому коді. Змінні, які використовуються в блоці коду, але пов'язані зі значенням поза коду, називаються вільними змінними.

Так як змінну можна пов'язати з об'єктом в будь-якому місці блоку, важливо, щоб це відбулося до її використання, інакше буде з’явиться виняток NameError. Зв'язування імен зі значеннями відбувається в операторах присвоювання, from, import, в формальних аргументах функцій, при визначенні функції або класу, в другому параметрі частини except оператора try-except.

Бажано, щоб програми не залежали від таких нюансів, а для цього досить дотримуватися наступних правил:

- Завжди слід пов'язувати змінну зі значенням до її використання.
- Необхідно уникати глобальних змінних і передавати все в якості параметрів. Глобальними на рівні модуля повинні залишитися тільки і константи, імена класів і функцій.
- Ніколи не слід використовувати from модуль import * - це може привести до затінення імен з інших модулів, а всередині визначення функції просто заборонено.

Краще переробити код, ніж використовувати глобальну змінну. Звичайно, для програм, що складаються з одного модуля, це не так важливо: адже всі визначені на рівні модуля змінні глобальні.

Прибрати зв'язок імені з об'єктом можна за допомогою оператора del. В цьому випадку, якщо об'єкт не має інших посилань на нього, він буде знищений. Для управління пам'яттю в Python використовується підрахунок посилань (reference counting), для видалення наборів об'єктів з зацикленими посиланнями - збірка сміття (garbage collection).