З чого складається програма

Отже, ще раз по порядку. З чого складається правильна програма?

Підключення модулів

Поки опускаємо подробиці, але ви вже бачили 2 модуля, які можна підключити до програми: sys та math. Це деякі готові бібліотеки, які надають вам доступ до якогось додаткового функціоналу і розширюють ваші можливості при написання програми. Їх підключення виглядало як:

```
import sys
import math
```

Будь-яка програма починається з підключення модулів. Можливо не цих, а якихось інших (не в будь-якій програмі потрібні математичні функції). Можливо вам не потрібне розширення функціоналу і ця секція програми просто залишиться порожньою. Але, якщо ви щось підключаєте, намагайтеся робити це на самому початку.

Визначення змінних

Перше, що, зазвичай, наявне в коді програми, -- визначення змінних. Python є досить гнучким і дозволяє вам створювати змінні будь-де в коді. Але гарний стиль оформлення програм передбачає, що змінні, по можливості, оголошуються в одному місці -- це, як мінімум, зручніше для програміста, який одразу бачить, які змінні є і яких вони типів.

Тут же змінним задаються початкові значення. Якщо конкретні значення не відомі або не потрібні на початку роботи, можна присвоїти спеціальне значення None -- "нічого". Тобто не нуль, не одиниця, не порожній рядок, а взагалі відсутність значення.

Всі змінні бажано називати змістовними іменами, щоб побачивши їх далі в коді,

можна було одразу зрозуміти, що вони роблять. За домовленістю змінні іменуються малими літерами (зверніть увагу: для python'а регістр важливий, тобто з точки зору інтерпретатора, наприклад, a і A -- різні змінні). Якщо одного слова для змістовної назви недостатньо, слова в імені змінної розділюються підкресленням.

Отже, string_length, variables_number, counter і навіть den_narodzhennia_olega -- хороші назви для змінних. DenProgramista, counter-of-loops -- погані. my_variable, asd1 -- дуже погані.

На жаль, вибрати змінній ім'я, яке повністю відображає її суть, можна не завжди. І тут нам на допомогу приходять коментарі.

Коментарі

Коментар -- це будь-який текст для пояснення роботи програми, який не виконується інтерпретатором. Коментарі можуть бути короткі: відокремлюються від основного коду дієзом і все, що знаходиться в коді програми після дієза і до кінця рядка, інтерпретатором ігнорується. Або довгі, в декілька рядків: перед початком багаторядкового коментаря ставиться троє подвійних лапок, і ще троє в кінці -- все, що знаходиться між ними, інтерпретатор також не виконає.

```
number = 10 # number of variables
"""
Here the program begins. This piece of text will be ignored while executing the
program
"""
number = number + 1
```

Крім тлумачення змінних коментарі можуть застосовуватися будь-де в коді програми для пояснення, що там відбувається. Часто буває, що, повертаючись через деякий час (тижня чи двох достатньо), автор вже не пам'ятає, для чого виконувалася та чи інша дія. А, якщо код програми мають читати сторонні люди (наприклад, програма розробляється кількома програмістами), коментарі тим більше не будуть зайвими.

Оголошення функцій

Так вийшло, що у нашому прикладі випадково з'явилася функція. І не готова, вбудована в python, а справжня користувацька, описана прямо в коді програми.

```
def text_prompt(msg):
  try:
    return raw_input(msg)
  except NameError:
    return input(msg)
```

Ви вже бачили математичні функції з модуля math, функції перетворення типів даних і ще кілька. Всі вони працювали за одним і тим самим принципом -- ви передаєте в них якісь дані, вони щось з тими даними роблять і повертають якийсь результат. Python дозволяє крім використання готових писати і власні функції, але поки не буду на цьому зупинятися -- це тема наступних лекцій. Втім, якщо в програмі є власні функції, вони оголошуються саме на початку -- після оголошення змінних або навіть перед ним.

Наша text_prompt виводить текстовий рядок-запрошення і чекає, поки користувач введе якісь дані і натисне Enter, а потім повертає введене значення в програму. На відміну від вводу з командного рядка, який розглядався на минулому занятті, це відбувається прямо під час роботи програми. А отже, за необхідності, ми можемо таким чином інтерактивно вводити дані в необхідні моменти часу.

Надалі вам вся ця складна конструкція непотрібна. Можете використовувати для цього функцію raw_input() (або input() -- для адептів python3). Що насправді робить text_prompt -- це просто забезпечує виконання raw_input або input незалежно від версії встановленого інтерпретатора. Отже, можна зробити просто отак:

```
x = raw_input('input x : ')
y = raw_input('input y : ')
```

```
summ = x + y
```

І, після того, як зроблено всі підготовчі оголошення, ми плавно перейшли до основного тіла програми.

Тіло програми

На початку тіла програми зазвичай відбувається введення даних. Це може бути інтерактивний запит raw_input(), як ми щойно розглянули, зчитування аргументів командного рядка, читання з якогось файлу або будь-який інший спосіб чи їх комбінація.

Коли всі необхідні дані отримано, можна проводити самі обчислення, для яких пишеться програма. Ця частина коду безпосередньо визначається алгоритмом та задачею, яку ви розв’язуєте, і давати якісь поради складно.

Метою ж будь-якої програми є перетворення деяких даних. Тобто ви вводите в програму певні значення, вона щось із ними робить і видає результат. Отже цей результат роботи в кінці має бути якимось чином виведено: збережено у файл, відправлено електронною поштою і т.д. Найпростіший варіант, який постійно використовується в цьому курсі, -- це просто вивести результат на екран, наприклад, командою print.

В літературі та прикладах ви можете зустріти різні варіанти запису print -- з дужками та без. Не переймайтеся, у python 2.7 обидва є вірними і ви можете використовувати їх в своєму коді:

```
print 'Hello!'
print('My name is Boris.')
```

Приклад з факторіалом

Розглянемо трохи вдосконалений приклад з факторіалом, додавши до нього докладні коментарі:

```
"""
Factorial calculation program
"""
n = 0                       # n to calculate the n!
counter = 0                 # counter = 1..n
result = 1                  # result = n!

def text_prompt(msg):       # function used for interactive data input
  try:
    return raw_input(msg)   # this one is for python 2
  except NameError:
    return input(msg)       # this one is for python 3

n = int(text_prompt('input N: '))          # input n
if n < 0:                                  # if n<0 then n! is undefined
  print('N should not be less than zero')
else:                                      # if n>=0 then calculate it
  for counter in range(n):                 # repeat n times
    result = result * (counter + 1)
  print(str(n) + '! = ' + str(result))     # print the result
```

На початку програми відбувається оголошення та ініціалізація змінних (присвоєння їм початкових значень), далі оголошення функції для введення даних та саме тіло програми, яке включає введення n, його перевірку, проведення обчислень та виведення результату. В даному прикладі замість того, щоб вивести просто число, ми виводимо цілий рядок, який складається з кількох частин, і виглядатиме при роботі програми якось так:

```
3! = 6
```

або

```
10! = 3628800
```

В принципі, це має сенс, якщо передбачається інтерактивна робота програми: ви виводите користувачеві рядок-запрошення, вводите дані, а далі виводите гарно відформатовану відповідь -- чому б і ні.

Таке форматування в даному випадку досягається конкатенацією рядків. Спочатку береться число n, яке залежить від того, що було введено користувачем. Зверніть увагу: так як додавання рядка і числа в python є недопустимим, ми примусово конвертуємо n в рядок за допомогою функції str(). До нього приписуємо окличний знак та дорівнює -- ця частина є незмінною при кожному запуску програми, тож просто записується як рядок. Далі до них додається власне результат, який також є числом, отже для додавання до рядка його теж необхідно конвертувати.

Крім форматування у даному прикладі замість додаткової змінної і використано count, а змінна n одразу вводиться як ціла, без зайвих перетворень.

Аналогічну програму можна було б записати без всякого інтерактиву, передаючи їй єдиний параметр -- число n -- під час запуску. Відповідно, в такому варіанті “дружнього” форматування відповіді теж можна позбавитись. І це виглядатиме приблизно наступним чином:

```
"""
Factorial calculation program
"""
import sys

n = 0                       # n to calculate the n!
counter = 0                 # counter = 1..n
result = 1                  # result = n!

n = int(sys.argv[1])        # input n
if n < 0:                   # if n<0 then n! is undefined
  print('no answer')
else:                       # if n>=0 then calculate it
  for counter in range(n):  # repeat n times
```

```
    result = result * (counter + 1)
  print(result)             # print the result
```

Майже те саме, тільки змінено введення та виведення даних, додано підключення модуля sys для читання аргументів командного рядка і, відповідно, прибрано функцію text_prompt -- вона більше не потрібна.

Алгоритмічні конструкції

Ключовим у прикладі є застосування алгоритмічних конструкцій -- умовного розгалуження та циклу Обидві вони передбачають нелінійне виконання програми, для чого можуть мати вкладені блоки коду -- саме ті фрагменти програми, які необхідно виключити з лінійної послідовності виконання дій в програмі.

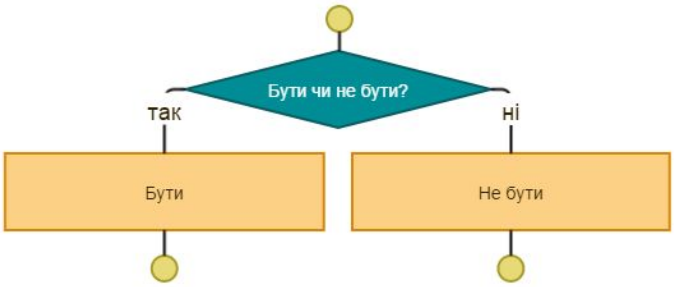
Нагадую: вкладені блоки коду в python виділяються відступами на початку рядка. Алгоритмічні структури можуть вкладатися одна в одну скільки завгодно разів -- це призводить до більшої вкладеності блоків коду і, відповідно, більших відступів.

```
if n == 0:
  print "n is equal to zero"
else:
  if n<0:
    if n == 7:
      print "You're lucky!"
    else:
      print 'n is less than zero'
  else:
    print "n is greater than zero"
```

З іншого боку це значить, що та частина коду, яка не передбачає вкладеності, тобто виконується послідовно, не має містити зайвих відступів, плаваючи вліво-вправо -- всі команди мають бути гарно написані на одному рівні.

Більшість мов програмування є досить гнучкими в цьому питанні, але містять додаткові оператори або позначки для виділення вкладених блоків. Розробники ж python пішли іншим шляхом: позбавились таким чином зайвих елементів у коді і одночасно змусили програмістів гарно форматувати код.

Умовне розгалуження -- if..else



Логічно, описує умову, за якої відбувається перехід на одну гілку чи іншу, та включає 2 вкладених блоки коду:

```
if n == 0:
  print "n is equal to zero"
else:
  print "n is not equal to zero"
```

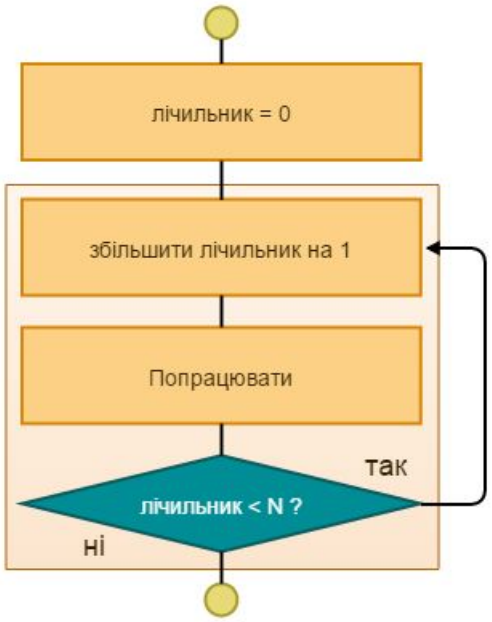
Якщо умова справджується, виконується перший блок. Якщо ні -- другий. Іноді буває, що необхідно щось робити лише у випадку, коли умова вірна -- в такому випадку другу частину конструкції можна опустити:

```
if n == 0:
  print "n is equal to zero"
print "this will be printed anyway"
```

В якості умови найчастіше застосовуються порівняння змінних із значеннями або іншими змінними. Для цього вам доступні <, >, <=, >=, ==. Для перевірки “не дорівнює” є аж два позначення: != та <>. Зверніть увагу: для перевірки, чи є значення або змінні рівними використовується подвійне “дорівнює” ==, одинарне ж застосовується для присвоєння значень.

Цикл for

Як ви побачили в Блоклі, використаний у прикладі for є лише одним із багатьох, але поки працюватимемо саме з ним. Це цикл із наперед визначеною кількістю ітерацій, у вигляді блок-схеми його можна представити приблизно так:



Не заглиблюючись у подробиці, ви можете підставити будь-яке N та повторити необхідну дію або послідовність дій (блок коду) N разів. При цьому змінна-лічильник циклу набуватиме на кожній ітерації значення від 0 до N-1. Наприклад, ми можемо вивести перші N натуральних чисел:

```
n = raw_input('input N:')
for counter in range(n):
  print counter+1
```

Або скласти між собою 10 членів арифметичної прогресії a1+a2+...+a10 з a0=1 та d=5:

```
a_0 = 1                    # zero member
a_previous = a_0           # previous member
a_i = 0                    # current member
```

```
d = 5
sum = a_0
for counter in range(10):
  a_i = a_previous + d     # calculate the new member
  sum = sum + a_i
  a_previous = a_i         # save it as previous
print a_i
```

Тестування

Розробка програми -- це не лише складання алгоритму та написання коду. Як наприкінці, так і в процесі постійно необхідно перевіряти працездатність написаного.

Дуже рідко програма працює лише з одним набором вхідних даних. Тобто попередній приклад із арифметичною прогресією -- скоріше виключення. А в більшості випадків для роботи вводяться довільні вхідні дані, які перетворюються під час роботи програми на вихідні. В ідеальному світі було б добре перевірити програму на всіх можливих вхідних значеннях і впевнитися, що все обчислюється вірно. Але навіть одна-єдина цілочисельна змінна на вході відповідає нескінченній кількості можливих значень.

Тому для перевірки роботи програми готується деякий набір тестів, які більш-менш повно представляють різні види можливих вхідних даних. Наприклад, логічно припустити, що, якщо наша програма вірно обчислює 2!, 3! та 4!, то з 5!, 6! і т.д. у неї також не виникне проблем. Це буде основним тестом для програми: декілька (2-3) типових значень з діапазону, на якому найчастіше використовуватиметься програма -- фактично, кілька чисел (або наборів чисел), які першими спадуть на думку.

Якщо ви знаєте, що в алгоритмі наявні важливі розгалуження, слід підбирати тестові дані таким чином, щоб протестувати кожен із можливих варіантів. Також в алгоритмі можуть бути присутні виключні ситуації, коли програма має

поводити себе нестандартно -- наприклад в тестах, які викликають ділення на нуль при обчисленнях.

Інша потенційна проблема -- це робота з дуже великими та дуже малими (близькими до нуля) значеннями. Обробка від'ємних чисел може відрізнятися від обробки додатних. Все це також має бути перевірено.

Мова поки не йде про ваші практичні завдання в рамках курсу, але у випадку серйозної програми, обов'язково мають бути випробовані недопустимі значення, програма має адекватно на них реагувати. Наприклад, для факторіалу це від'ємні та дійсні числа, рядки.

Керуючись цими принципами та знаннями про будову програми можна скласти порівняно невеликий набір тестів, який, тим не менш, буде досить повно покривати різні ситуації, які можуть виникати під час роботи програми. З його допомогою можна як перевірити правильність готової програми, так і відстежувати порушення в її роботі при подальшому удосконаленні, розширенні функціональності та ін.

Аналогічно складаються тести для проміжних перевірок в процесі розробки, лише часто тестується не програма в цілому, а окремі її вузли.