Python – да се хвърлим в програмирането

Crash course into programming with Python for ages 11-17

Съдържание:

Необходими знания преди началото на курса

Предполага се, че учениците имат богат опит с графичен език за програмиране от типа на Scratch или LEGO NXT-G, знаят какво е MyBlock или функция, знаят и какво е променлива и как се прави елементарен брояч. Също така са свикнали при показване на текст да превръщат числата от числов тип (Number) в текст.

Предварителна работа и основни знания

Инсталиране на Python

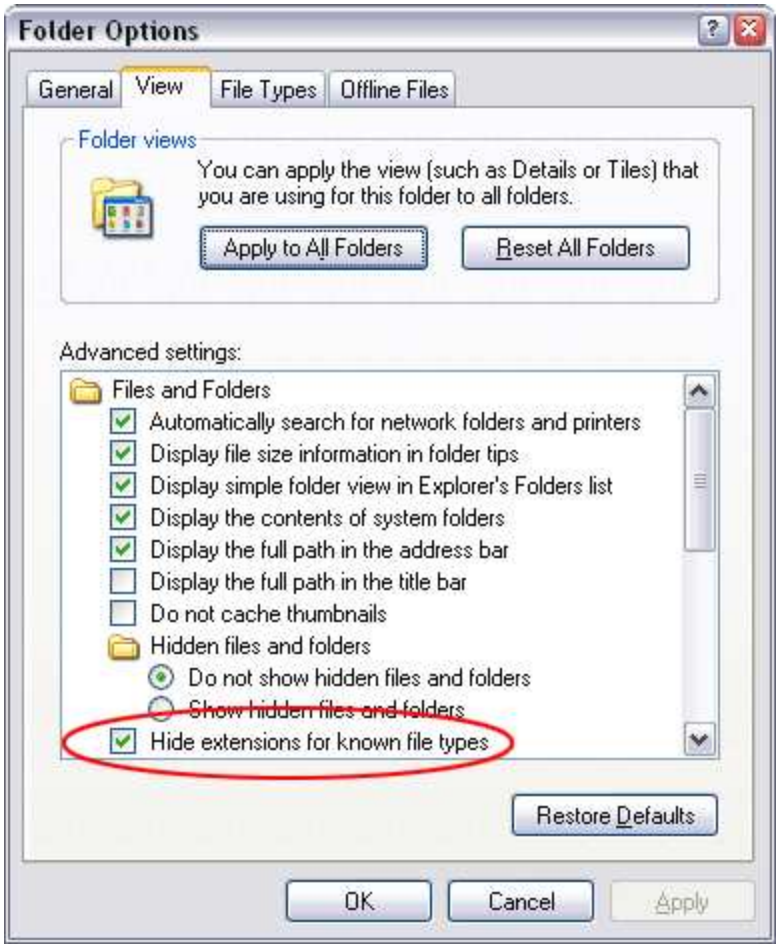
В този курс ще използваме Python, поради високата му лекота за работа.

Изтеглете и инсталирайте Python от http://www.python.org/getit/

Настройка на Windows да показва разширенията на файловете

Отворете произволна папка и в менюто *Tools* изберете *Folder Options...*

В появилият се диалог изберете таб-а *View*, намерете полето *Hide extensions for known file type* (*Скриване на разширенията за познати типове файлове*) и **махнете** отметката на полето.



Стартиране на текстовия редактор за нашите програми

Изберете си папка, в която ще се намират вашите програми и щракнете на празно място в нея с десен клавиш на мишката, изберете New > и изберете Text Document. Като име на файла задайте proba.py. Разширението *.py* означава, че създаваме програма (по точно скрипт) на езика Python.
Двойното щракване просто изпълнява програмата, която към този момент е празна.
За да можем да пишем във файла или за да го редактираме трябва да щракнем с десен клавиш *върху него* и да изберем *Edit with IDLE*.
Отваря се текстовият редактор IDLE и в него напишете на първия ред, като текстът ще се оцвети сам:

```
print("Vsichko raboti kakto trqbva")
```

Запишете като натиснете клавишната комбинация от клавиша Control и клавиша S, от тук нататък ще се изписва като Ctrl+S.

Използване на конзолата на Windows

Натиснете Ctrl+R, което ще ви покаже диалога *Run...*
Напишете, че искате да изпълните програмата *cmd*. Отваря се прозорец на Command Promt и той се намира в папка:
C:\Documents and Settings\[вашето_потребителско_име] > _
Този ред се чете като: “Намираме се в папка *[вашето_потребителско_име]*, която се намира в папка *“Documents and Settings”*, която от своя страна се намира в дял *C:*”

Сега трябва да промените работната папка.

Командата, която ви трябва се нарича *cd*, съкратено от ChangeDirectory, като се употребява по някой от следните два начина:

1. За да влезем в папка, която се намира в такущата, например папка Desktop:

C:\Documents and Settings\[вашето_потребителско_име] > cd Desktop
C:\Documents and Settings\[вашето_потребителско_име]\Desktop > _

2. За да се върнем една папка назад по пътеката:

C:\Documents and Settings\[вашето_потребителско_име]\Desktop > cd ..
C:\Documents and Settings\[вашето_потребителско_име] > _

3. С наклонени черти можем да минаваме през няколко папки на веднъж:

C:\Documents and Settings\[вашето_потребителско_име] > cd ..\..
C:\ > _
C:\ > cd “Documents and Settings \ [вашето_потребителско_име] \ Desktop \ Python “
C:\Documents and Settings\[вашето_потребителско_име]\Desktop\Python > _

4. Клавишът *TAB* спестява печатането на имената на папките. Използвайте го смело и на воля!

Изпълняване на програмата

След като сте стартирали Command Prompt отидете в папката си за програми на Питон, например:

```
C:\Documents and Settings\[потребителско_име] > cd Desktop\Python
```

и там напишете името на програмата, която искате да изпълните в случая ***proba.py***

```
C:\Documents and Settings\[потребителско_име]\Desktop\Python > proba.py
```

и резултатът е:

```
C:\Documents and Settings\[потребителско_име]\Desktop\Python > proba.py
Vsichko raboti kakto trqbva
C:\Documents and Settings\[потребителско_име]\Desktop\Python > _
```

Програмата изпечатва текста в конзолата и приключва. Всичко е настроено и работи както трябва!

Знанията за и от урок №1 - променливи и условия

Променливи и работа с тях

Създаване на променливи

Променливи се създават като се напише име за променливата и на нея се присвои стойност със знака за равенство “=”. Важно е името на променливата да е говорещо и да няма празни места в него.
Началната стойност на променлива може да е число, текст, или друга променлива, като например:

```
ime_promenliva_za_chislo = 5
vtoro_chislo = 10
treto_chislo = vtoro_chislo
```

В резултат на тези инструкции третото число ще стане равно на второто, като второто вече е равно на 10.

Работа с променливи

Променливите, които съдържат числа могат да изпълняват всички математически действия по между си, като резултатът е най-удобно да се записва в нова променлива:

```
rezultat_1 = vtoro_chislo + treto_chislo # пример за събиране на променливи
```

Колко ще е резултатът от събирането на двете числа?

```
rezultat_2 = treto_chislo * 4 # умножение на число и променлива
```

Свободни сме да запишем резултата от операциите и в същата променлива ако желаем. Ето пример за намаляване на стойността на променлива на половина:

```
treto_chislo = treto_chislo / 2 # делението се прави с дробна черта
```

Можете ли да напишете сега програма, която създава променлива със начална стойност сто и след това намалява променливата с единица, като записва резултата в същата променлива.

Недефинирана променлива

Ако се опитате да работите с променлива, която все още не сте създали ще получите грешка:

```
treto_chislo = peto_chislo - 2
peto_chilso = 15
```

NameError: name 'peto_chilso' is not defined

Можете да поправите грешката, като създадете (иначе казано дефинирате) променливата peto_chislo преди реда, на който ще я използвате:

```
peto_chilso = 15
treto_chislo = peto_chislo - 2
```

Другата често допускана грешка е да си мислите, че сте дефинирали променливата, но сте допуснали грешка при писането:

```
dyljina = 15
shirina = daljina * 2 # а правилното е dyljina * 2
```

Ако пуснете програмата ще получите грешката:

NameError: name 'daljina' is not defined

Писане на коментари

Може би сте забелязали, че червеният текст след инструкциите служи да поясни какво се случва на дадения ред. Всичко след знака # (‘диез’) няма да се изпълни от компютъра и служи на човека пишещ програмата да си води записки къде какво се случва.
Коментари могат да се слагат и на самостоятелни редове.
Следват три примера за една и съща програма, преценете сами коя е най-ясна.

Без коментрари и с **лошо** именуване на променливите:

```
a=10
h=2
s=a*h/2
```

Със коментари и **лошо** именуване на променливите:

```
a = 10 # страна а
h = 2 # височина
s = a*h/2 # лице
```

Без коментари, но с *описателни* имена на променливите:

```
strana_a = 10
visochina = 2
lice_na_triygylnik = strana_a * visochina / 2
```

Забележете как допълните интервали около знаците за равенство, умножение и деление улесняват четенето на кода.

Печатане на текст и променливи в конзолата

В началните настройки вече направихме програма, която печата конкретен текст:
`print(“Vsichko raboti kakto trqbva”)`
и резултатът ще е
Vsichko raboti kakto trqbva

Освен текст можем да разпечатаме и стойността на променлива:
`print (visochina)`
и резултатът ще е
2

Най-полезно ще ни е ако можем да покажем описателен текст на потребителя и към него да долепим стойността на променлива. Слепването на два текста се прави със знак +
`print (“Vsichko raboti” + “kakto trqbva”)`
и резултатът ще е
Vsichko rabotikakto trqbva
като трябва да се забележи, че слепването не слага интервали между текстовете, които слепва. Ето защо по-красив резултат ще се получи от реда:
`print (“Vsichko raboti ” + “kakto trqbva”)`
където след думата raboti има празно място.

Превръщане на число в текст

Слепването на текст и променлива работи по подобен начин:
`print (“Liceto na triygylnika e ” + lice_na_triygylnik)`
Това обаче ще ни даде грешка ако се опитате да го изпълните:
TypeError: cannot concatenate 'str' and 'int' objects

Програмата има нужда от изрична инструкция за превръщане на числовата променлива (integer, int) във низ от знаци (string, str).
Смисълът на това превръщане е, че числата се съхраняват само в една клетка памет в компютъра, докато за текстовете е нужна по една клетка за всяка буква. Ето защо числото хиляда например, се помни в една клетка, но се изписва с четири символа - “1000” и има нужда от превръщане от число към текст.

В Python това превръщане се прави с вградената функция *str(),* която се извиква със числото или променливата, което искаме да превърнем:
`print (“Liceto na triygylnika e ” + str(lice_na_triygylnik) “)`
Това вече ще покаже текущата стойност за лицето:
Liceto na triygylnika e 10

И понеже лицето се измерва в квадратни сантиметри ще добавим още малко текст:

```
print (“Liceto na triygylnika e ” + str(lice_na_triygylnik) + “kv. cm”)
```

Liceto na triygylnika e 10kv.cm

Генериране на случайни числа

За да симулираме зар или друг случаен генератор използваме библиотеката *random*, която първо трябва да импортираме в нашата програма

```
import random
```

За истинското генериране на случайните числа от библиотеката *random* използваме функцията *randint()* която се извиква с два стойности - от кое минимално число и до кое максимално число да ни дава случайни числа.
За да създадем зарче в компютъра можем да напишем:

```
import random
hvyrlqne_na_zar = random.randint(1, 6)
print (“Zara se padna na chisloto ” + str(hvyrlqne_na_zar) )
```

Можете ли да направите теглене на тото в компютъра? Напишете програма която изтегля 6 случайни числа между 1 и 49.

Въвеждане на число от потребителя на програмата

По време на първия урок ни се наложи да попитаме потребителя на програмата за число. Това се постига с функцията *input*() на която се подава текстът, който да запита потребителя за число:

```
input(“Vyvedete chislo:”)
```

Vyvedete chislo:_

Важно е да запомните че функцията *input*() винаги връща текст, и ако ви трябва да получите число от нея, трябва да превърнете текста в число.
Това се постига чрез функцията *int*(), която приема текст.

Постоянно в програмите ще виждате фунцията *input*() поставена вътре във функцията *int*(), за да може въведеното да се превърне в число и да може да се използва по-нататък в програмата:

```
vyvedeno = int(input(“Vyvedete chislo:”))
```

Набирате число и натискате клавиша Enter. Обикновено искаме запомним числото, което е въведено, ето защо трябва да го запишем в променлива:

```
vyvedeno = int(input(“Vyvedete chislo:”))
dvoyno = vyvedeno * 2 # този ред се изпълнява чак щом се натисне Enter
print (“Udvoeno to e ravno na ” +str(dvoyno))
```

Vyvedete chislo:_
1128
Udvoeno to e ravno na 2256

Проверки на условия

Смисълът на програмирането е програмите ви сами да преценяват условията, които вие опишете. Условията взимат моментните стойности на променливите в програмата и проверяват дали условието е истина.

Условие IF (ако, дали)

Нека да проверим дали човекът е въвел числото едно:

```
vyvedeno = int(input("Vyvedete chislo:"))
if vyvedeno == 1:
    print ("Natisnali ste 1")
print ("Kray na programata")
```

Ще изпълня програмата и няма да въведа 1:

```
Vyvedete chislo:_
11
Kray na programata
```

Изпълни се само четвъртия ред на програма, условието в if-а не е изпълнено и print-ът на третия ред не се изпълни.

Сега вече ще въведа 1:

```
Vyvedete chislo:_
1
Natisnali ste 1
Kray na programata
```

Условие ELSE (иначе, в противен случай)

Ако има нужда можем да хванем всички случаи които не са хванати от if-а:

```
vyvedeno = inint(input(“Vyvedete chislo:”))
if vyvedeno == 1:
   print (“Natisnali ste 1”)
else:
   print (“Drugo neshto ste natisnali, dobre.”)
print (“Kray na programata”)
```

Ако въведа 1, ще имаме точно същия резултат като в предишния случай. Можете ли да предположите ако изпълня тази програма и въведа 22?

```
Vyvedete chislo:_
22
Drugo neshto ste natisnali, dobre.
Kray na programata
```

Видове проверки

Експериментирайте със следните условия:

```
if vyvedeno != 1: # проверка за “различно”

if vyvedeno > 0: # проверка за положително число

if vyvedeno < 10: # проверка за едноцифрено число

if vyvedeno >= 10: # проверка за “по-голямо или равно” на 10
```

Правила при използване на if/else

- Редът на който е написан if (или else) завършва с двуеточие - ":"
  Ако няма двуеточие ще получите синтактична грешка:

```
        if vyvedeno == 1
                       ^
SyntaxError: invalid syntax
```

- Отместването на кода след if (или else) трябва да е по-навътре от самият if/else:

```
# код от основната линия на програмата
if условие:
    # код, който се изпълнява в if
    # и може да е от няколко реда
else:
    # код, който се изпълнява в else
    # и може да е от няколко реда
# код от основната линия на програмата,
# който се изпълнява независимо,
# от това как са минали проверките
```

Отместването на кода (indentation)

В Python отместването на кода с няколко интервала определя коя инструкция при какви условия се изпълнява. Обмислете внимателно следния код, който дава пример за вложени условия:

```
vyvedeno = int(input("Vyvedete chislo:"))
if vyvedeno > 10:
   print ("Vyveli ste neshto nad deset")
   if vyvedeno == 20:
      print (", po-tocno 20")
   if vyvedeno > 99:
      print (" i to daje ne e dvucigreno")
else:
   print ("Vyveli ste neshto ednocifreno")
print ("Kray na programata")
```

Какво ще се разпечата на екрана ако въведете 1, 11 или 111?

Възможните съобщения за грешка свързването с отместването, които може да получите са:

- За излишно отместване:

```
vyvedeno = int(input("Vyvedete chislo:"))
   print ("Blagodaria")
```

Грешка:

```
  print "1"
  ^
IndentationError: unexpected indent
```

- За липсващо изместване:

```
vyvedeno = int(input("Vyvedete chislo:"))
if vyvedeno == 1:
print ("Natisnali ste 1")
```

Грешка:

```
print "Natisnali ste 1"
      ^
IndentationError: expected an indented block
```

Знанията за и от урок №2 - цикли и мрежово програмиране

Цикли

Понякога се налага част от кодовите инструкции да се повтарят по няколко пъти. Броят повторения може да ни е известен предварително, а може и да не знаем колко пъти ще се повтори цикълът.

Точно определен брой повторения

Да речем, че искаме да изтеглим 5 числа от тотото:

```
import random
for i in range (0, 5):
    print random.randint(1, 35)
```

Резултатът при мен беше:

```
12
30
34
10
26
```

Използваме цикъл *for* в който създаваме променлива ‘*i*’ (името идва от индекс), който се повтаря 5 пъти.
Петте повторения се определят от числото 5 в израза *range (0, 5)*.
Във *for*-овете също важи правилото за отместването, описано при if-else.

Безкраен цикъл и условие за излизане

В други случаи искаме програмата да работи вечно, или докато не се изпълни някакво условие.

Програма която никога няма да спре от самосебеси се прави по следният начин:

```
import random
while True:
    print (random.randint(1, 35))
```

Ако се объркате да пуснете тази програма имате няколко варианта:

- Спирате тока вкъщи (има опасност от шамари)
- Спирате тока на компютъра (има опасност да се развали)
- Затваряте конзолата (трябва да пишете cd Desktop\Python отново)
- Натискате Ctrl+C докато се изпълнява програмата (препоръчителен вариант)

Ако искаме да печатаме случайни числа от едно до сто и да спрем щом се падне точно сто, можем да напишем следното условие вътре във *while* цикъла:

```
import random
while True:
    r = random.randint(1, 100)
    print r
    if r == 100:
        break
```

Break

Инструкцията и думата break означават ***прекрати*** и спират *най-вътрешния* цикъл в който са вложени.

Мрежово програмиране

Интернет се състои от програми-сървъри, които предоставят информация, например съдържанието на уеб-сайтове и програми-клиенти, които се обръщат към тези сървъри и обработват получената информация, като пример е браузърът, с който влизате в “интернет”.
В този урок учениците ще се научат да създват програми за клиент и за сървър.

Терминология

IP адрес - адрес по интернет протокол, на клиента или сървъра.

Порт - един компютър има 65535 мрежови порта, по които може да си говори с други устройства. За дадена услуга сървърът слуша на един порт, на който да се свързват клиентите му.

Сокет (socket) - виртуален контакт свързващ клиент и сървър. Намира се в библиотеката *socket*.

Изпращане - процесът на изпращане на данни по сокет, независимо дали от или сървър, което трябва да бъде прието от отсрещната страна. Подобно на функцията *input()*, изпращането не преминава на следващия ред от програмата, докато не бъде прието. Изпращане на данни се прави с функцията *send()* като данните трябва да се превърнат в текст преди да бъдат изпратени.

Приемане - процеса на изчакване до получаване на информация по сокет-а. Програмата не продължава на следващия ред докато не получи нещо по мрежата. Данните, които се получават винаги са от тип текст, а функцията за това е *recv()*.

Минимален сървър

Нека да създадем сървър, който първо поздравява и после връща като ехо това, което му изпратим.

```
# програма за ехо-сървър
import socket
host = ""
port = 22122
kanal = socket.socket()
kanal.bind((host, port))
# следва още
```

Първо импортираме библиотеката *socket*, създаваме променливи за хост и порт, като хост се състои от празен текст, две кавички една до друга. Ние сме сървъра и няма нужда да оказваме адрес.
Портът може да е което пожелаете число, най-добре да е над 1024, и ако получавате грешки опирайте да смените порта.
След това създаваме канал за комуникация с фукнцията *socket()* от библиотеката *socket*. След това обвързваме канала за комуникация с нашия адрес и порта, който сме избрали чрез функцията *bind()* за създадения вече канал.

След това сървърът започва да слуша по канала за клиенти искащи връзка с функцията *listen()*. Щом такъв клиент се появи програмата преминава на следващия ред, където приема клиента с фунцията *accept()* и на веднъж записва две различни неща в две променливи.

По осъществената връзка сървърът ни изпраща поздрав “Hello!” чрез функцията send() и след това започва да чака да получи нещо от клиента чрез *recv()*, като иска да получи не повече от 1024 байта на веднъж (един килобайт, дори по-точно един кибибайт).

Щом получи данните, ги записва в променлива и преминава на следващия ред, къде изпраща на обратно това, което е получил. Малко като програма за папагал. Щом повтори обаче, каквото е получил, сървърът затваря връзката чрез close() и започва цикълът отначало, като отново чака някой да се свърже с него. Вижте в началото на урока как се спира безкраен цикъл.

```
# продължава отгоре
while True:
    print ("waiting ...")
    kanal.listen(1)
    vryzka, adres = kanal.accept()
```

```
    print (“Connected by “ + str(adres))
    
    vryzka.send("Hello!")
    danni = vryzka.recv(1024)

    vryzka.send(danni)
    vryzka.close()
    print ("closed")
#край на програмата
```

Минимален клиент

След като имаме сървър можем да напишем и клиент, който да се свърже с него и да провери дали работи правилно.

```
# клиент за ехо сървъра
import socket
host = “localhost”
port = 22122
vryzka = socket.socket()
vryzka.connect((host, port))
# следва още
```

Има само две разлики в тази част от кода на клиента и сървъра. Първата е в хост-а, който тук е *localhost* т.е. вашия собствен компютър. Втората е в начинът, по който клиентът се свързва със сървъра - вместо *bind()* използва функцията *connect()*.

По-нататък е важно клиентът да следва протокола на сървъра. Протокол означачава правилата, по които си говорят две програми. В нашият случай протоколът е:

1. Сървърът изпраща поздрав - клиентът трябва да приеме поздрава
2. Сървърът очаква съобщение от клиента - клиентът трябва да го изпрати
3. Сървърът изпраща съобщение на клиента - той трябва да го приеме
4. Връзката се затваря

Така че, първата работа на клиента, след успешно осъществяване на връзка е да получи поздрава на сървъра. Това става с познатата функция *recv()* която записва в променлива, която за всеки случай изпечатваме на екрана.
Щом получим поздрава на сървъра трябва да минем към изпращане на нещо. Тук просто му изпращаме друг поздрав чрез *send()*, но можете да помолите потребителя да въведе число и да изпратите него на сървъра.
Щом сме изпратили “Здравейте” към сървъра е време да видим какво ще ни изпрати той. отново получаваме данни с *recv()*, записваме ги в друга променлива, показваме я на екрана и затваряме връзката.

```
#продължава от горе
pozdrav = vryzka.recv(1024)
print ("pyrvo poluchih " + pozdrav)

vryzka.send("Zdraveyte")

danni = vryzka.recv(1024)
print ("posle poluchih " + danni)

vryzka.close()
#край на програмата
```

Показаният пример е илюстративен и можете да измислите свой протоколи на разговор. Сървърът не е длъжен да поздравява и може да остави първото изпращане за клиента. Изпращанията и приеманията може да не се редуват, а може да имате няколко от един вид в която посока желаете.
Можете да имате втори цикъл във сървъра, който да приключва връзката когато клиентът пожелае, а както е в този пример.

Числа по мрежата

Уловката в тази ситуация е, че ако се опитате да изпратите число с функцията *send()* ще получите грешка:

```
    vryzka.send(15)
TypeError: must be string or buffer, not int
```

За да поправите грешката погледнете урок едно, печатане на променливи и текст и превръщането на число в текст (иначе казано string).

Другата уловка е при приемането. Представете си, че изпратите успешно числото 15 от по-горе. И поискате да го разделите на две и да го върнете на клиетна. Задачата с отдалечененото пресмянате е доста актуална.
Следният код ще даде грешка:

```
chislo = vryzka.recv(1024)
udvoeno = chislo / 2
```

```
    udvoeno = chislo / 2
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

Грешката казва, че се опитваме да делим променлива от тип текст. Както пише в началото на урока за мрежовото програмиране, всичко което минава по мрежата е текст.
Ето защо първо трябва да превърнете текста в число с функцията *int()*.

```
chislo = int(chislo)
```

Знака за деление работи добре ако му подадете числа, но хвърля грешки ако му подадете букви.

Множество изпращания по ред

Представете си следната програма:

```
vryzka.send("Zdraveyte")
vryzka.send("Zdraveyte")
```

и приемникът от другата страна:

```
vryzka.recv(1024) # ще получил ZdraveyteZdraveyte
vryzka.recv(1024) # би чакал безкрайно
```

Оказва се, че ако изпратите две неща в два последователни send()-а, те може да бъдат приети на веднъж от един recv().

За да заобиколим проблема можем да използваме малка пауза между пращанията от библиотеката *time:*

```
import time
# програма, програма
vryzka.send("Zdraveyte")
time.sleep(1) # пауза от една секунда
vryzka.send("Zdraveyte")
```

В този случай ще заобиколите проблема, че вторият ви *recv()* чака безкрайно.

Използвана литература

1. http://www.tutorialspoint.com/python/
2. http://www.fileinfo.com/help/windows_show_extensions
3. http://docs.python.org/3.0/library/socket.html

```
"""
GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
Permissions beyond the scope of this license may be available at
http://www.robopartans.com
"""
"""
Idea
 The server is general with *no* game logic inside, so that it serves for *any* game with numbers.
 That forces all game logic to be in the clients, which for in case is good for education.
 That also allows clients to cheat, which is *great* for education.

Workflow
1) start with dialogs for number of rooms and players per room.
2) then accept clients one by one and ask them for room number
3) then send the client his player number in the room

4) once the room is full the  server ask all players in order for their numbers.
5) once the numbers queue is full  the server send to all clients all numbers with 0.1s delay.
6) repeat step 4

7) if client send "" or number less than 0 the connection with him and the whole game room is closed
"""

import socket
import thread
import time
import random

class RoomServer:

  def __init__(self):
    self.num_rooms = input("How many gamerooms do you need?\n")
    self.players_per_room = input("How many players per room do you wish?\n")

    print (" = = = = = = = = = = = = = = = = = = = =")
    print ("Robopartans Gamerooms Server is started!")
    print (" = = = = = = = = = = = = = = = = = = = =")

    self.gameRooms = []
    for i in range (0, self.num_rooms):
      self.gameRooms.append([])

    host = ''
    port = 22122
    s = socket.socket()
    s.bind((host, port))

    while True:
      print "awaiting connections..."
      s.listen(1)
      conn, addr = s.accept()
      print 'Connected by ' +str(addr)
      self.accept_client(conn)


  def accept_client(self, conn):
    conn.send("Hello, player! Choose gameroom from 1 to "+str(self.num_rooms))

    data = conn.recv(1024)
    print "Player chose room #" + data
```

```
    self.assign_player_to_room(int(data)-1, conn)


  def assign_player_to_room (self, room_number, conn):
    self.gameRooms[room_number].append(conn)
    print "entered room #"+ str(room_number+1)

    players_in_curr_room = len(self.gameRooms[room_number])

    conn.send("your player ID in the room is:")
    time.sleep(0.1)
    conn.send(str(players_in_curr_room))

    if players_in_curr_room == self.players_per_room:
      players = self.gameRooms[room_number]
      self.gameRooms[room_number] = [] # clear the list item and prepare to accept new set of players in the
same room number

      room_id = hex(random.randint(0x100,0x999))[2:5] # random 3 symbol sequence
      print "room #"+str(room_number) +" is now known as "+room_id

      thread.start_new_thread(gameRoom, (room_id, players))


def gameRoom(room_id, players):
  """one thread per game room"""

  print "room #"+ room_id+ " starts playing"
  work = True

  while work:
    numbers_list = []

    #ask each player for new number
    for conn in players:
      plr_id = "P"+str(players.index(conn)+1)

      conn.send("[room "+room_id+"]: "+plr_id+", send me your number.")
      data = conn.recv(1024)

      if data:
        if int(data) < 0: # our QUIT mesage
          print plr_id + " @" +room_id+ " sent QUIT"
          work = False
          break # for

        print plr_id + " @" +room_id+ " sent '" + data + "'"

        numbers_list.append(int(data))

      else:
        print plr_id + " @" +room_id+  " sent nothing."
        work = False
        break # the for loop I hope

    if work:
      #send the accumulated numbers to all clients
      for conn in players:
        for num in numbers_list:
          conn.send(str(num))
          time.sleep(0.1) # ! most important part, so that they get numbers(messages) one by one, not
concatenated
  #eof while

  #if not data: break
  for conn in players:
    conn.close()

  print "we closed room " + room_id
```

```
#Program start
RoomServer()
```

```
"""
GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
Permissions beyond the scope of this license may be available at
http://www.robopartans.com
"""
"""
This program implements and tests the gameRoomsServer.py Robpartans server.
"""

import socket
import random
import time

host = 'localhost'        # The remote host
port = 22122              # The same port as used by the server
s = socket.socket()
s.connect((host, port))


data = s.recv(1024) #receive greeting and question
print('Received', data)

#~ s.send(str(1))
s.send(str(random.randint(1, 2))) #send choice - room 1
print('sent input for room')

print s.recv(1024) # reveive player ID message

plr_id = int(s.recv(1024)) # receive actual ID
print "I am #"+str(plr_id)+" in the room"

for i in range(1, 5):

  data = s.recv(1024) #receive question for number
  print('Received Q:', data)

  #~ raw_input("press key to send random number ")
  time.sleep(random.randint(1, 4)) # emulate delay for automated testing
  s.send(str(random.randint(1, 6))) # send random dice
  print "sent dice"

  # receive all numbers, mine and others
  for i in range (1, 3):
    data = s.recv(1024)
    print('Received P'+str(i), repr(data))


#leave
print "sending -1"
s.send(str(-1)) # our QUIT message

print "closing"
s.close()
```