MVA Microsoft Virtual Academy

Learning to code with Python!

Microsoft

Microsoft
Virtual
Academy

String variables and asking a user to enter a value

input

How can we ask a user for information?

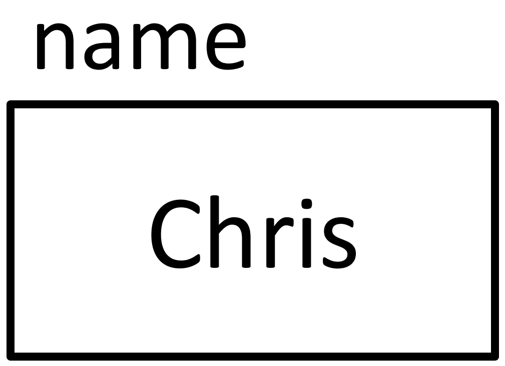
```
name = input("What is your name? ")
```

The **input** function allows you to specify a message to display and returns the value typed in by the user.

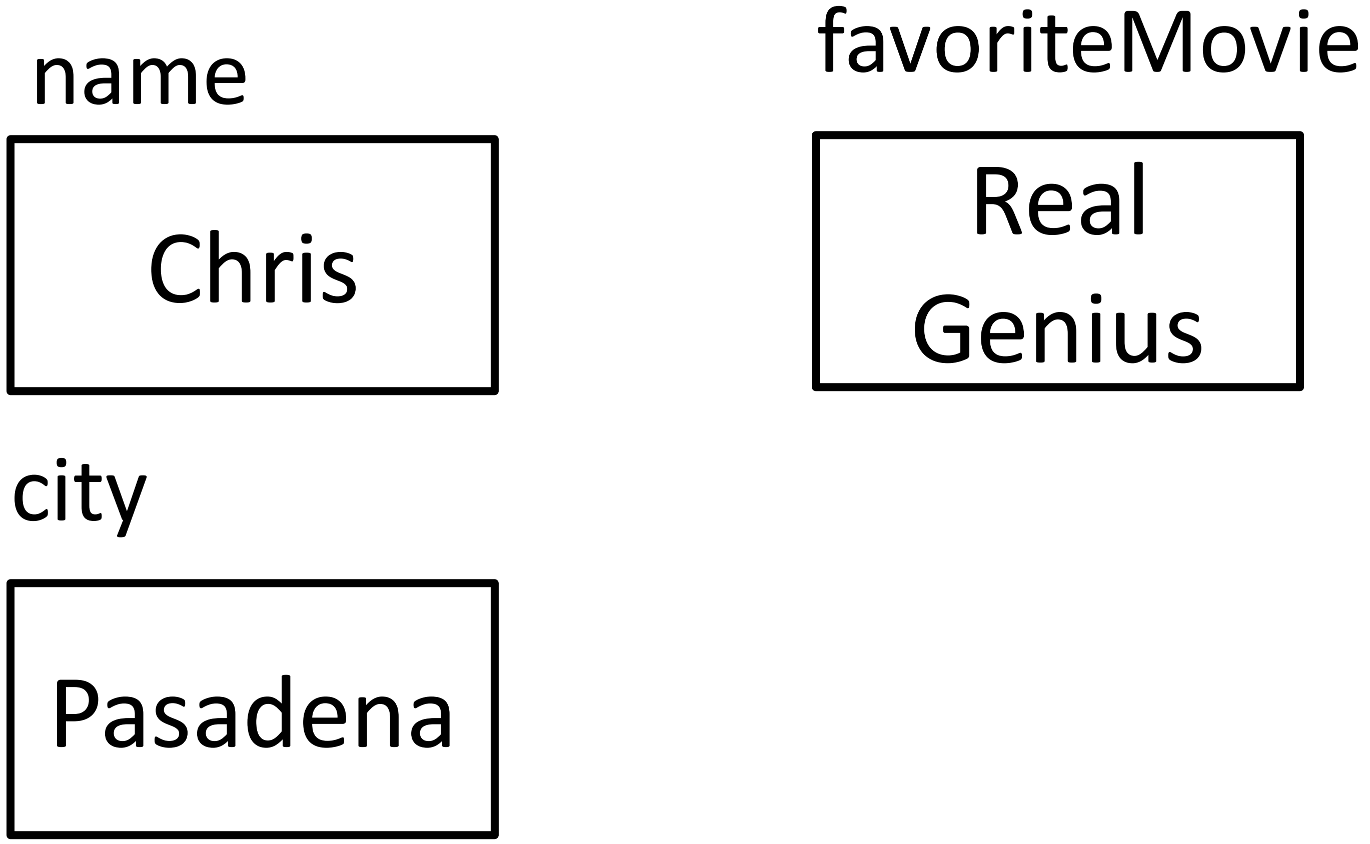
We use a variable to remember the value entered by the user.

We called our variable “name” but you can call it just about anything as long the variable name doesn’t contain spaces

Think of a variable as a box where you can store something and come back to get it later.

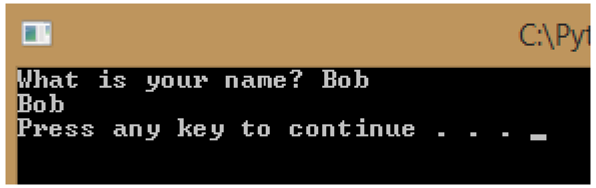


If you need to remember more than one value, just create more variables



You can access the value you stored later in your code

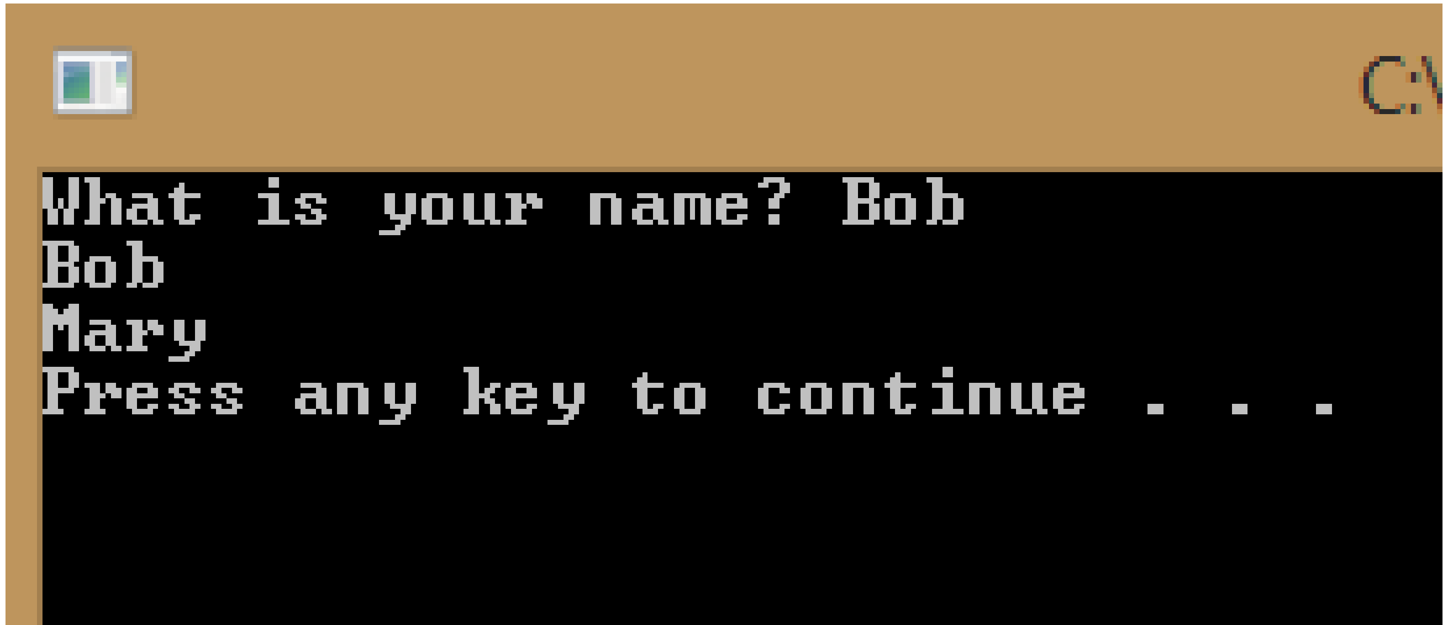
```
name = input("What is your name?
print(name)
                                    "
```



```
What is your name? Bob
Bob
Press any key to continue . . .
```

You can also change the value of a variable later in the code

```
name = input("What is your name?
print(name)
name = "Mary"
print(name)
```



```
What is your name? Bob
Bob
Mary
Press any key to continue . . .
```

Which of the following do you think would be good names for variables?

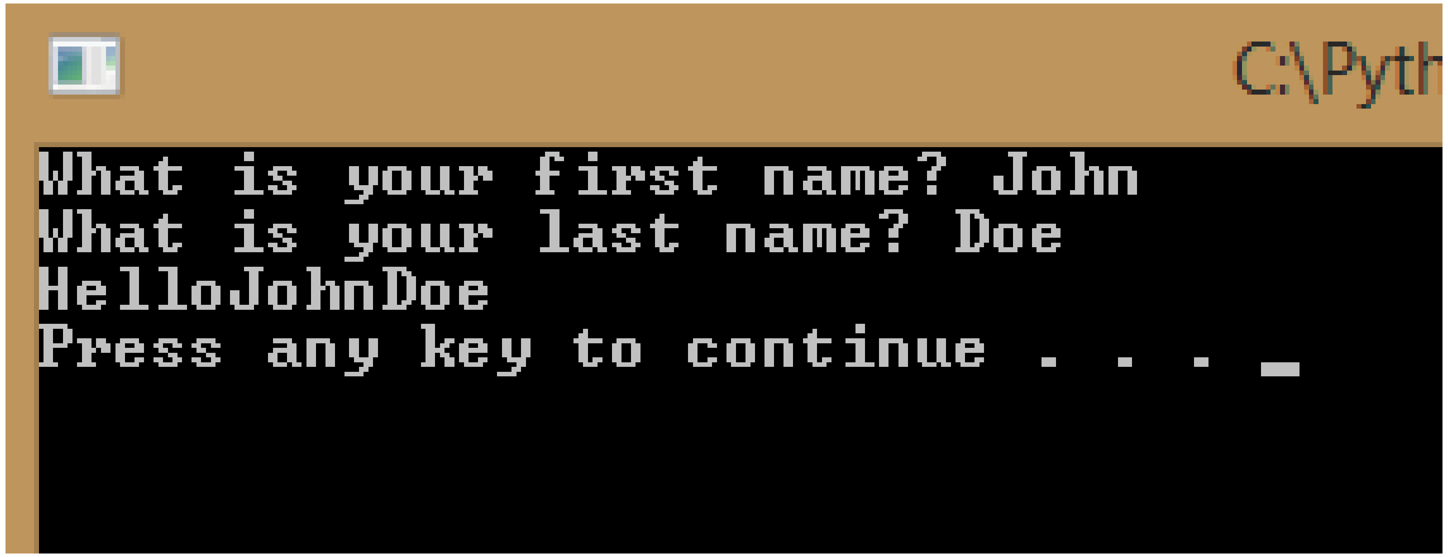
- Variable1
- First Name
- Date
- 3Name
- DOB
- DateOfBirth
- YourFavoriteSignInTheHoroscope

Variable names

- Should be meaningful (e.g. FirstName not variable1)
- Should be specific (BirthDate not Date)
- Should not contain spaces (FirstName not First Name)
- Should be descriptive but not too long (FavoriteSign not YourFavoriteSignInTheHoroscope)
- Are case sensitive (FirstName and firstname would be two different variables)
- Cannot start with a number (Name1 is okay 1Name is not)

You can combine variables and strings with the + symbol

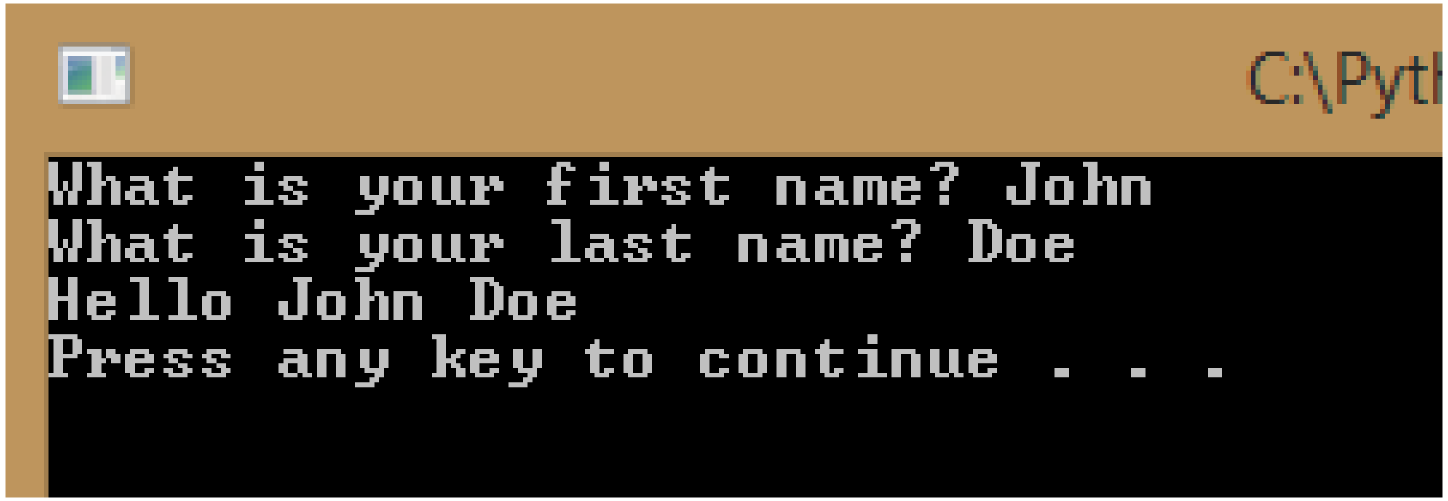
```python
firstName = input("What is your first name? ")
lastName = input("What is your last name? " )
print("Hello" + firstName + lastName)
```



```
What is your first name? John
What is your last name? Doe
HelloJohnDoe
Press any key to continue . . . _
```

Often you need to add punctuation or spaces to format the output correctly

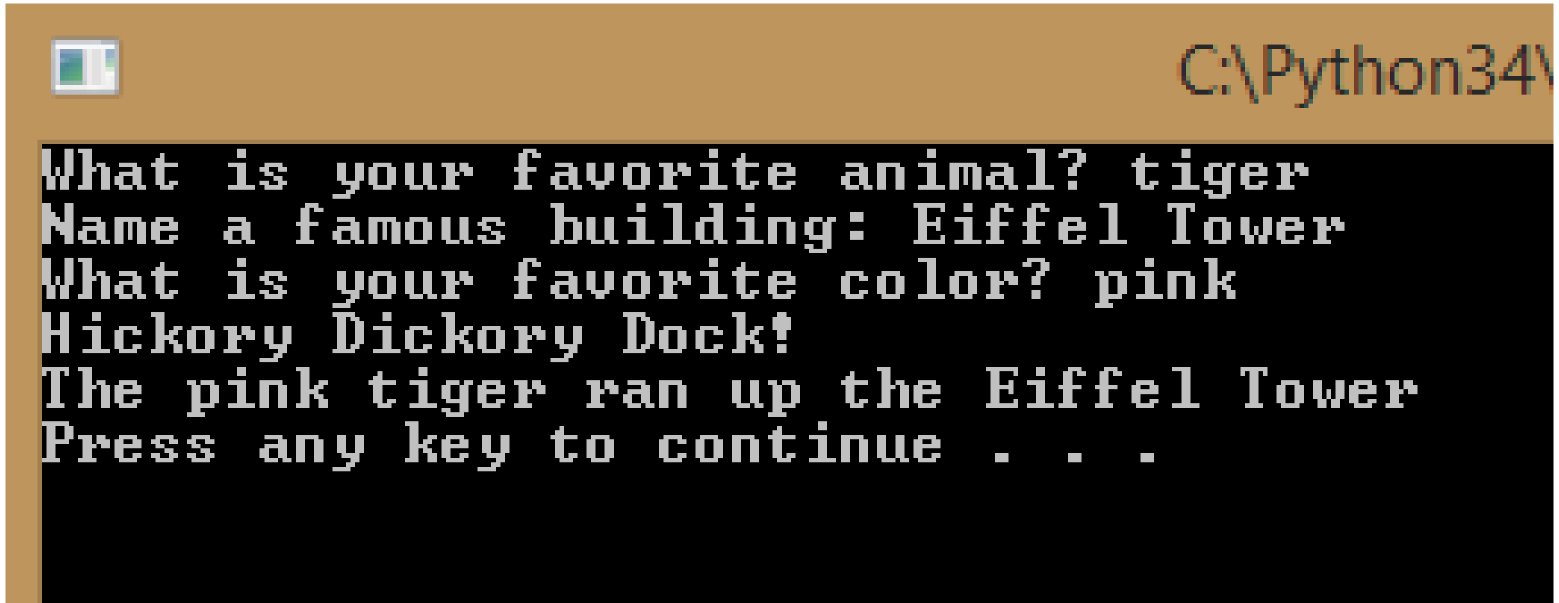
```
firstName = input("What is your first name? ")
lastName = input("What is your last name? " )
print("Hello " + firstName + " " + lastName)
```



```
What is your first name? John
What is your last name? Doe
Hello John Doe
Press any key to continue . . .
```

Now you can create a story teller program!

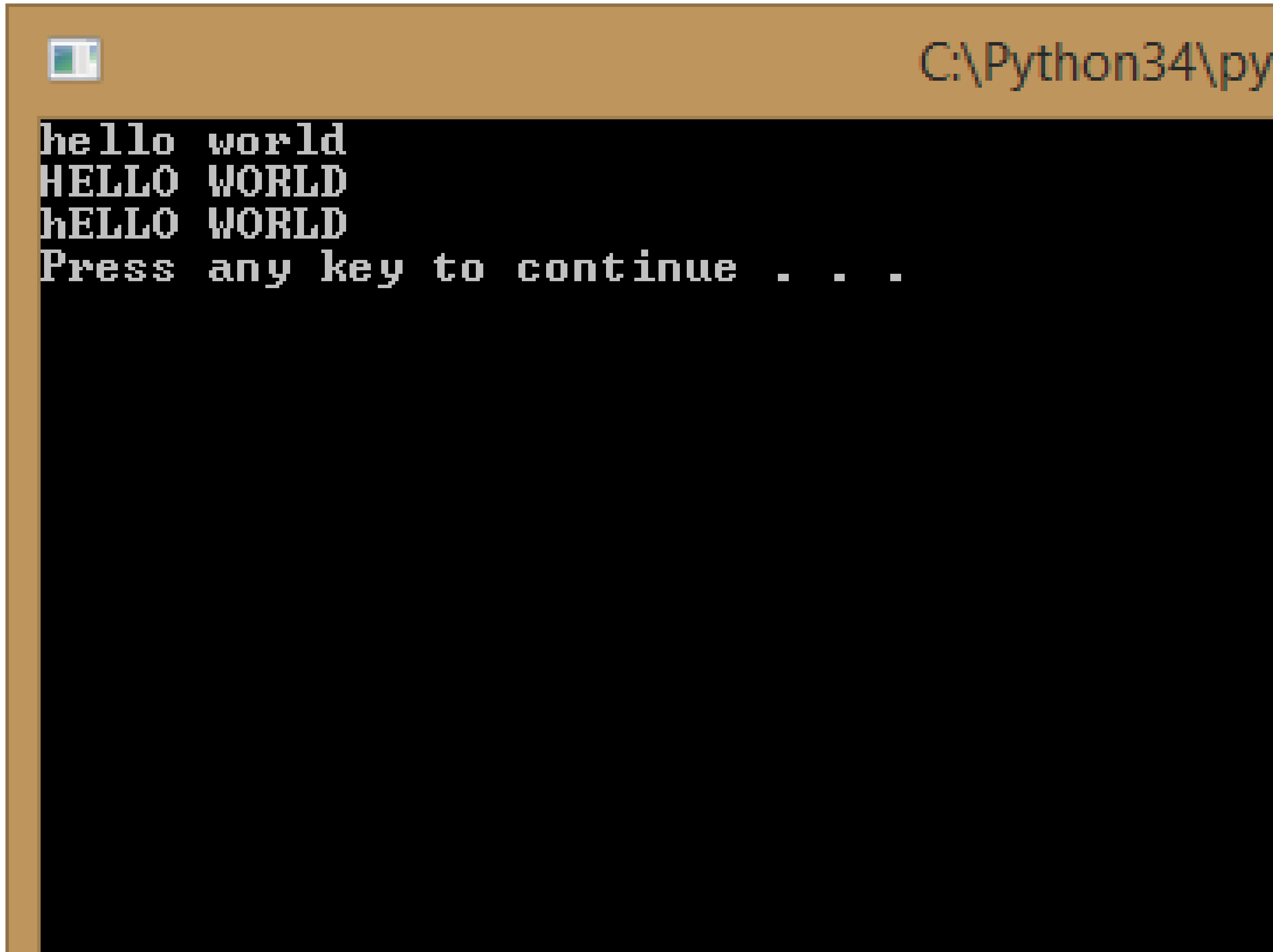
```python
animal = input("What is your favorite animal? " )
building = input("Name a famous building: ")
color = input("What is your favorite color? ")
print("Hickory Dickory Dock!")
print("The "+color+" "+animal+" ran up the "+building)
```



```
What is your favorite animal? tiger
Name a famous building: Eiffel Tower
What is your favorite color? pink
Hickory Dickory Dock!
The pink tiger ran up the Eiffel Tower
Press any key to continue . . .
```

Variables also allow you to manipulate the contents of the variable

```
message = 'Hello world'
print(message.lower())
print(message.upper())
print(message.swapcase())
```



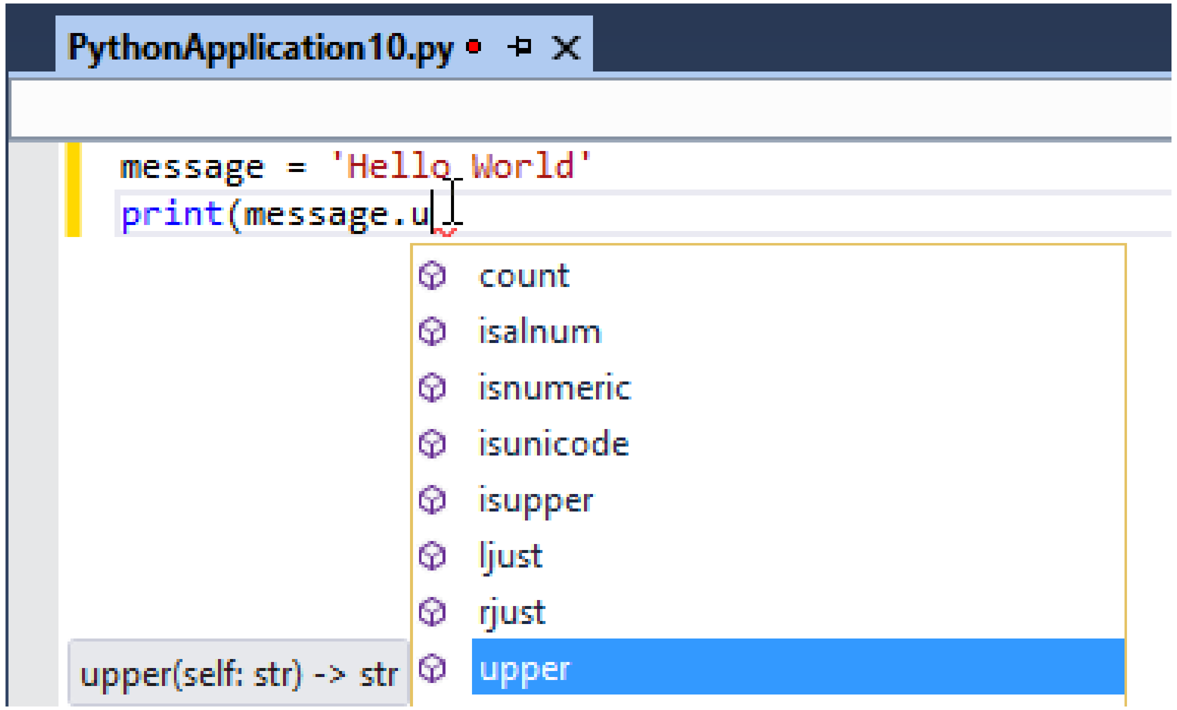
```
hello world
HELLO WORLD
hELLO WORLD
Press any key to continue . . .
```

Did you notice the pop up list?

That’s intellisense.

Visual Studio will suggest possible functions that you can call automatically after you type the ‘.’

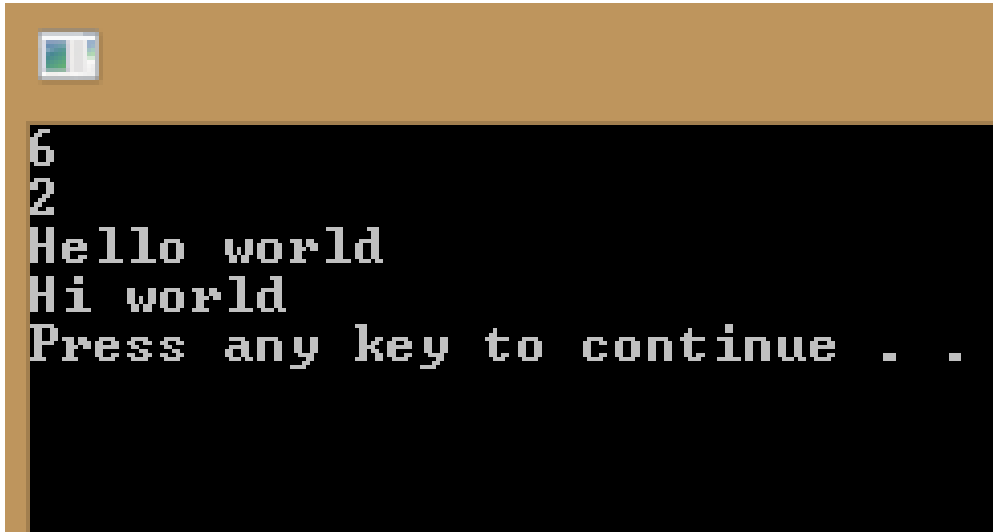
You can also use CTRL+J or CTRL+SPACE to launch intellisense



```
message = 'Hello World'
print(message.u
```

What do you think these functions will do?

```python
message = 'Hello world'
print(message.find('world'))
print(message.count('o'))
print(message.capitalize())
print(message.replace('Hello','Hi'))
```



```
6
2
Hello world
Hi world
Press any key to continue . .
```

Programmers do not memorize all these functions!!

So how do programmers find them when they need them?

- Intellisense
- Documentation
- Internet searches

How could we…

Have a user enter their postal code and then display that postal code in all upper case letters even if the user typed it in lowercase?

```
postalCode = input("Please enter your postal code: ")
print(postalCode.upper())
```

Did you notice?

The intellisense didn’t appear to help us select the **upper()** function.

That’s because our program didn’t know we were going to store a string value, in the postalCode variable. The **upper()** function is only for strings. A good habit when coding in any language is to initialize your variables. That means when you create them you give them an initial value.

```
postalCode = " "
postalCode = input("Please enter your postal code: ")
print(postalCode.upper())
```

How could we…

Ask someone for their name and then display the name someone with the first letter of their first and last name uppercase and the rest of their name lowercase?

```
name = " "
name = input("Please enter your name: ")
print(name.capitalize())
```

Functions and variables allow us to make new mistakes in our code…

```
Each line of code below has a mistake…
```

```
message = Hello world
23message = 'Hello world'
New message = 'Hi there'
print(message.upper)
print(mesage.lower())
print(message.count())
```

```
message = 'Hello world'
23message = 'Hello world'
New message = 'Hi there'
print(message.upper())
print(message.lower())
print(message.count('H'))
```

Storing numbers

You can store numbers in variables as well

```
age = 42
print(age)
```

With numbers you can do math

```
width = 20
height = 5

area = width * height
print(area)

perimeter = 2*width + 2*height
print(perimeter)

perimeter = 2*(width+height)
print(perimeter)
```

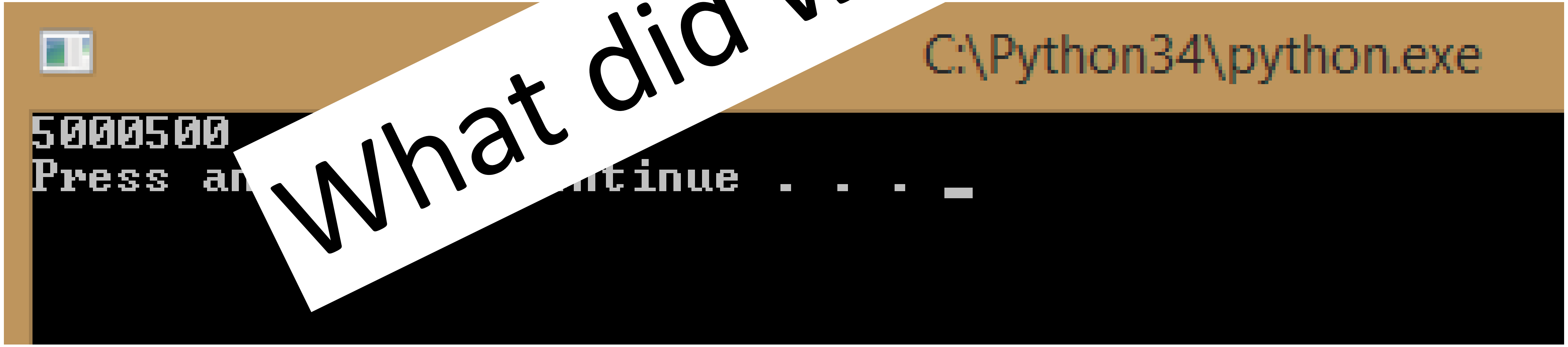
Math rules haven’t changed since Grade 4

```
Order of operations
( ) parentheses
**  exponent (e.g. **2 squared **3 cubed)
*/  multiplication and division
+ - addition and subtraction
```

Of course this means we have more ways to make mistakes too!

```
salary = '5000'
bonus = '500'
payCheck = salary + bonus
print(payCheck)
```

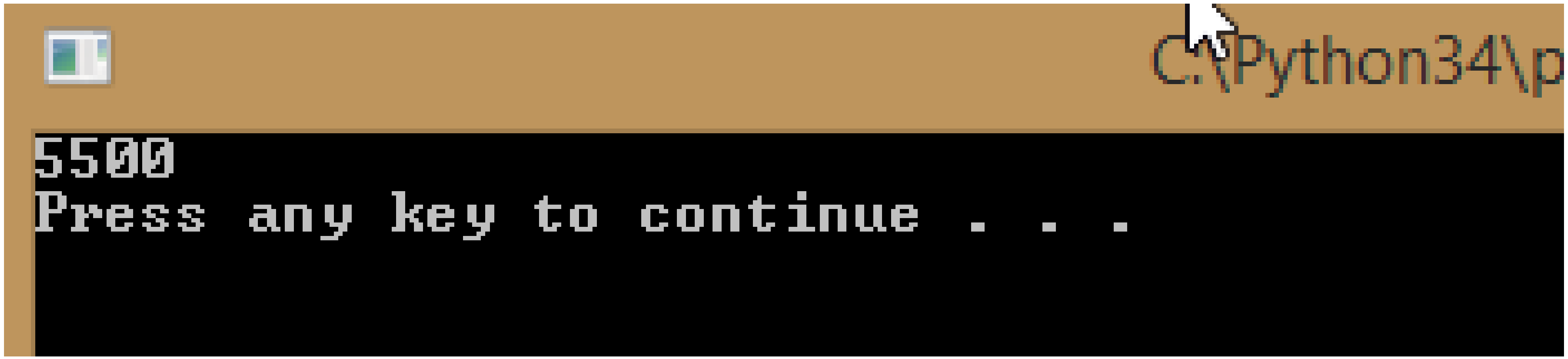
What did we do wrong?



```
5000500
Press an      ntinue . . . _
```

Because we put quotes around the values, the program thought salary and bonus were strings so it concatenated instead of adding

```
salary = 5000
bonus = 500
payCheck = salary + bonus
print(payCheck)
```

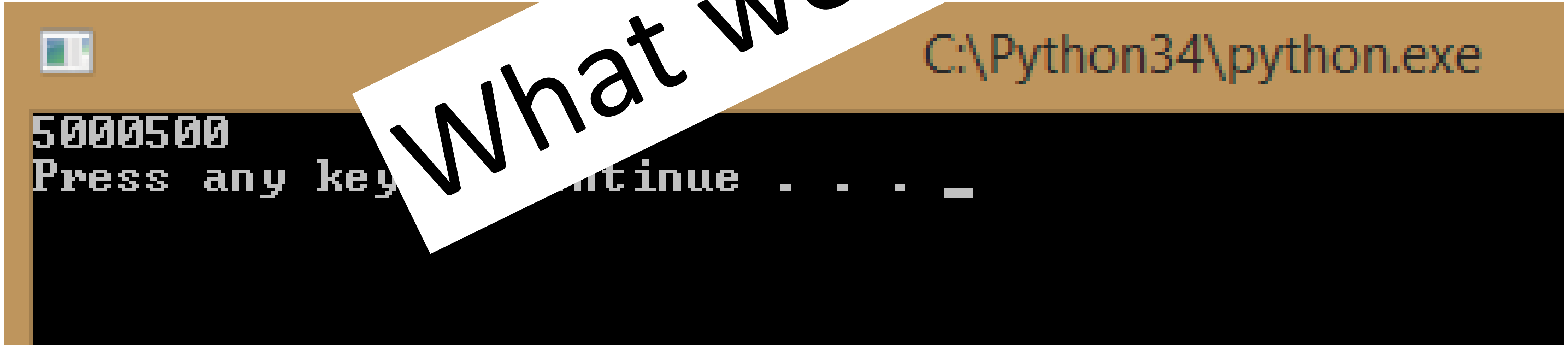


```
5500
Press any key to continue . . .
```

Could you ask the user to enter their bonus and salary values?

```
salary = input("Please enter your sal
bonus = input("Please enter your
payCheck = salary + bonus
print(payCheck)
```

What went wrong?



```
5000500
Press any key     ntinue . . . _
```

The input function always returns a string

We need a way to tell our program that it’s a number and not a string

There are functions to convert from one datatype to another.

```
int(value)    converts to an integer
long(value)   converts to a long integer
float(value)  converts to a floating number (i.e. a
number that can hold decimal places)
str(value)    converts to a string
```

Which function should we use for our scenario?

Since the amounts entered could include decimals – choose float

```
salary = input("Please enter your salary: ")
bonus = input("Please enter your bonus: ")
payCheck = float(salary) + float(bonus)
print(payCheck)
```

What do you think will happen if someone types “BOB” as their salary

The code crashes because we can’t convert the string “BOB” into a numeric value. We will learn how to handle errors later!

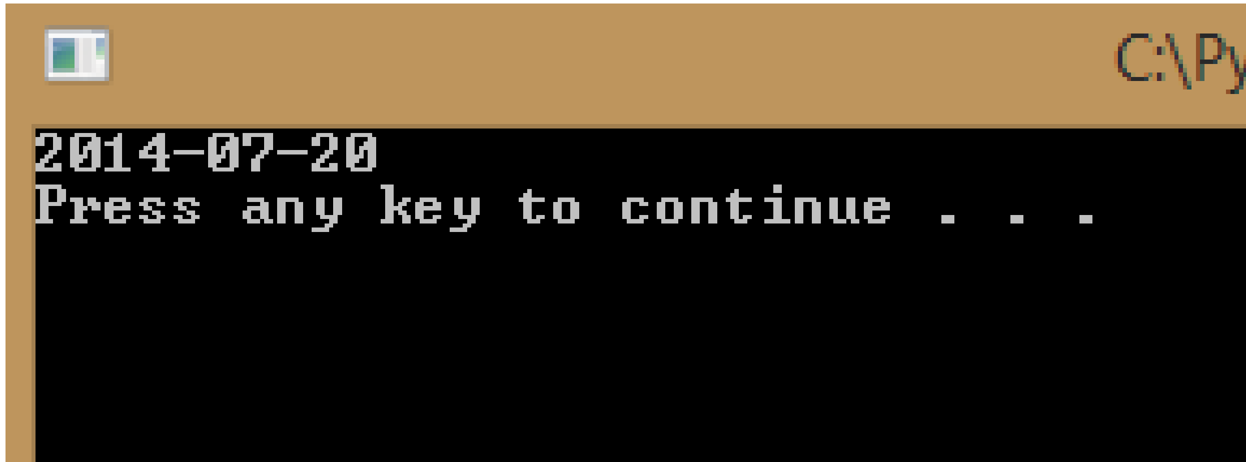
Working with dates and times

datetime

What is today’s date?

- The datetime class allows us to get the current date and time

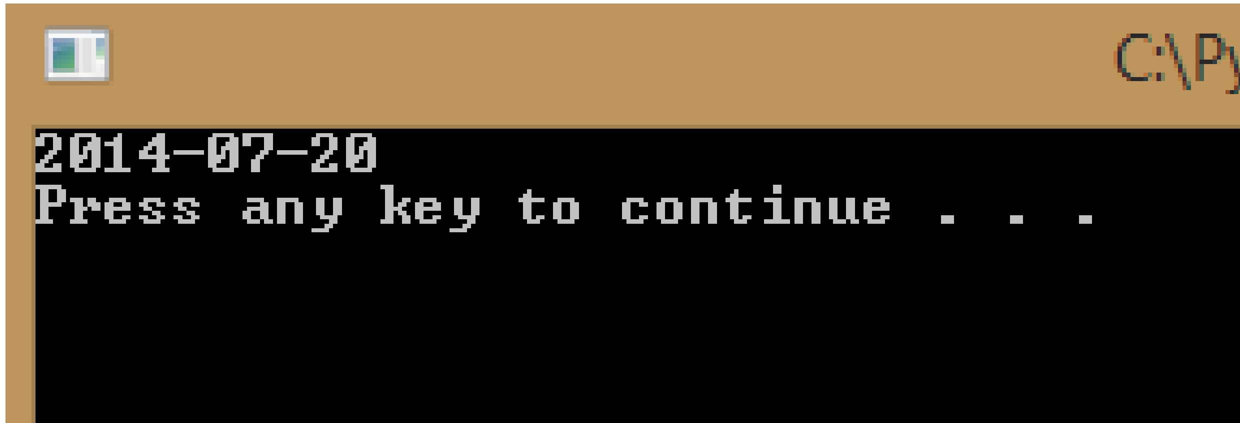
```
#The import statement gives us access to
#the functionality of the datetime class
import datetime
#today is a function that returns today's date
print (datetime.date.today())
```



```
2014-07-20
Press any key to continue . . .
```

You can store dates in variables

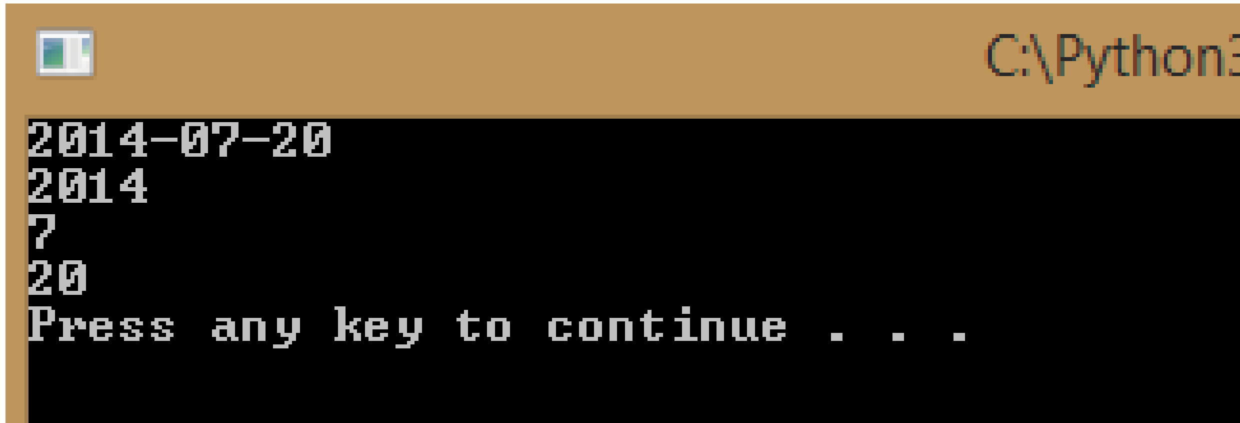
```
import datetime

#store the value in a variable called currentDate
currentDate = datetime.date.today()
print (currentDate)
```



```
2014-07-20
Press any key to continue . . .
```

You can access different parts of the date

```
import datetime
currentDate = datetime.date.today()
print (currentDate)
print (currentDate.year)
print (currentDate.month)
print (currentDate.day)
```



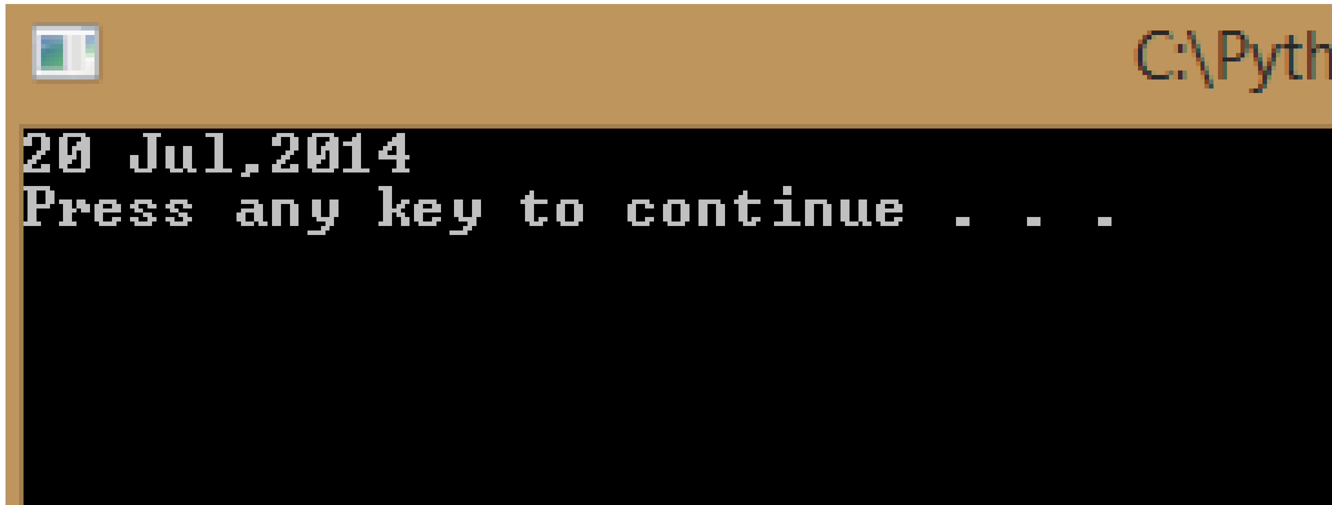
```
2014-07-20
2014
7
20
Press any key to continue . . .
```

But what if you want to display the date with a different format?

- Welcome to one of the things that drives programmers insane!
- Different countries and different users like different date formats, often the default isn’t what you need
- There is always a way to handle it, but it will take a little time and extra code
- The default format is YYYY-MM-DD

In Python we use strftime to format dates

```python
import datetime
currentDate = datetime.date.today()
#strftime allows you to specify the date format
print (currentDate.strftime('%d %b,%Y'))
```



```
20 Jul,2014
Press any key to continue . . .
```

What the heck are %d %b and %Y?

- %d is the day of the month
- %b is the abbreviation for the current month
- %Y is the 4 digit year

Here's a few more you may find useful

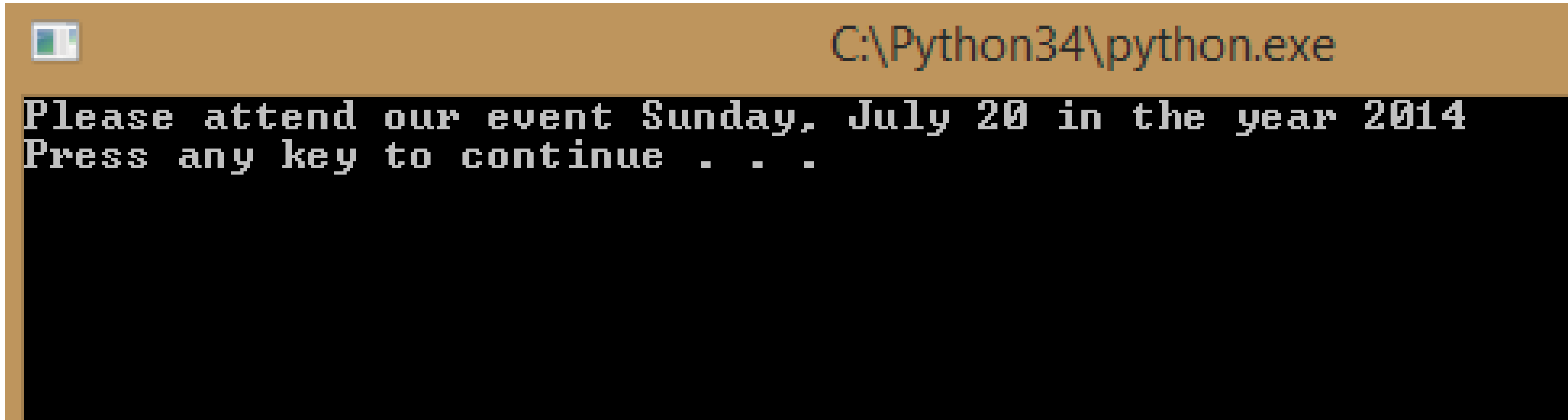
- %b is the month abbreviation
- %B is the full month name
- %y is two digit year
- %a is the day of the week abbreviated
- %A is the day of the week

- For a full list visit strftime.org

Could you print out a wedding invitation?

"Please attend our event Sunday, July 20th in the year 1997"

```python
import datetime
currentDate = datetime.date.today()
#strftime allows you to specify the date format
print (currentDate.strftime
('Please attend our event %A, %B %d in the year %Y'))
```



C:\Python34\python.exe

```
Please attend our event Sunday, July 20 in the year 2014
Press any key to continue . . .
```

So… what if I don't want English?

- In programmer speak we call that localization
- Did I mention dates drive programmers insane?
- By default the program uses the language of the machine where it is running
- But… since if you can't always rely on computer settings it is possible to force Python to use a particular language
- It just takes more time and more code. You will probably want the babel Python library http://babel.pocoo.org/

Can I ask a user for their birthday?

```
birthday = input ("What is your birthday? ")
print ("Your birthday is " + birthday)
```

Can you think of any situations where this code might not work the way we want?

Can I ask a user for their birthday?

```
birthday = input ("What is your birthday? ")
print ("Your birthday is " + birthday)
```

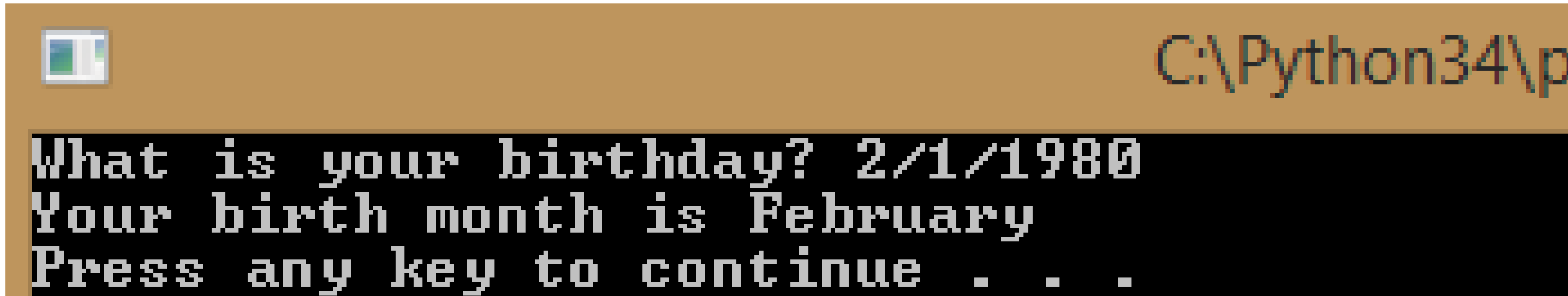
What datatype is birthday?

string

if we want to treat it like a date (for example use the datetime functions to print it in a particular format) we must convert it to a date

The strptime function allows you to convert a string to a date

```python
import datetime
birthday = input ("What is your birthday? ")
birthdate =
datetime.datetime.strptime(birthday,"%m/%d/%Y").date()
#why did we list datetime twice?
#because we are calling the strptime function
#which is part of the datetime class
#which is in the datetime module
print ("Your birth month is " + birthdate.strftime('%B'))
```



```
What is your birthday? 2/1/1980
Your birth month is February
Press any key to continue . . .
```

But what if the user doesn’t enter the date in the format I specify in strptime?

```
birthdate = datetime.datetime.strptime(birthday,"%m/%d/%Y")
```

- Your code will crash so…
- Tell the user the date format you want

```
birthday = input ("What is your birthday? (mm/dd/yyyy) ")
```

- Add error handling, which we will cover in a later module

Dates seem like a lot of hassle, is it worth it? Why not just store them as strings!

- You can create a countdown to say how many days until a big event or holiday

```
nextBirthday = 
datetime.datetime.strptime('12/20/2014','%m/%d/%Y').date()
currentDate = datetime.date.today()
#If you subtract two dates you get back the number of days
#between those dates
print (nextBirthday - currentDate)
```

Dates seem like a lot of hassle, is it worth it? Why not just store them as strings!

- You can tell someone when the milk in their fridge will expire

```
currentDate = datetime.date.today()
#timedelta allows you to specify the time
#to add or subtract from a date
print (currentDate + datetime.timedelta(days=15))
print (currentDate + datetime.timedelta(hours=15))
```

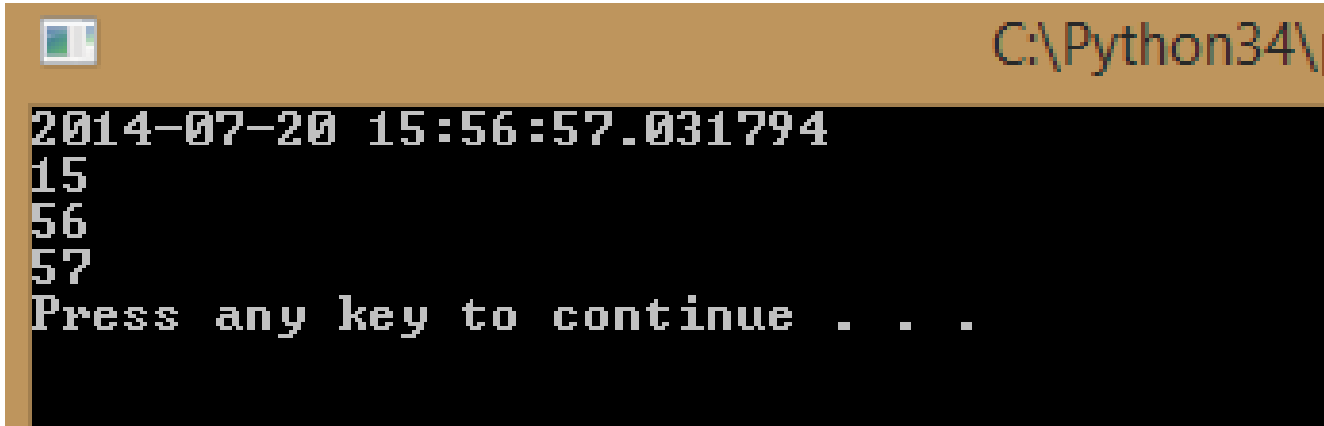
You will be amazed how often you need to work with dates!

- If datetime doesn’t have what you need, check out the dateutil library (for example you might want to know the number of years between two dates instead of number of days)

What about times?

- It is called Date**time**, so yes it can store times.

```
import datetime
currentTime = datetime.datetime.now()
print (currentTime)
print (currentTime.hour)
print (currentTime.minute)
print (currentTime.second)
```

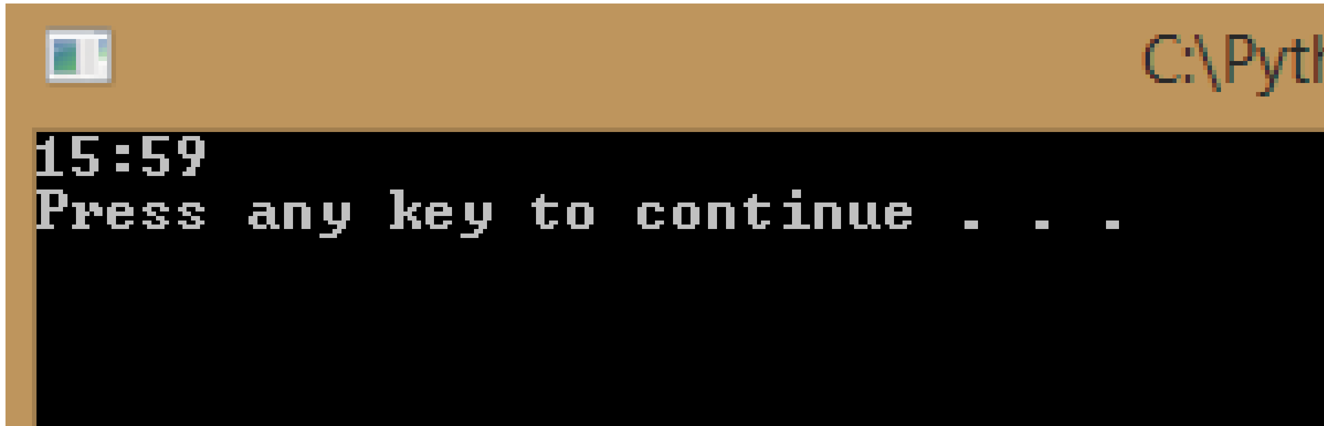


```
2014-07-20 15:56:57.031794
15
56
57
Press any key to continue . . .
```

Just like with dates you can use strftime() to format the way a time is displayed

```python
import datetime
currentTime = datetime.datetime.now()
print (datetime.datetime.strftime(currentTime,'%H:%M'))
```

%H	Hours (24 hr clock)
%I	Hours (12 hr clock)
%p	AM or PM
%m	Minutes
%S	Seconds



```
15:59
Press any key to continue . . .
```

MVA Microsoft Virtual Academy

Conditional code

Microsoft

If statements

Sometimes you want your code to react differently to different situations

- If the user chooses express shipping, charge an extra $10
- If I win the lottery quit my job
- If the hockey player gets the puck in the net, add one to the score

If statements allow you to specify code that only executes if a specific condition is true

```
answer=input("Would you like express shipping?")
if answer == "yes" :
    print("That will be an extra $10")
```

What do you think the == symbol means?

You can use different symbols to check for different conditions

```
==   is equal to                 if answer == "yes" :
!=   is not equal to             if answer !=  "no" :
<    is less than                if total < 100 :
>    is greater than             if total > 100 :
<=   is less than or equal to    if total <= 100 :
>=   is great than or equal to   if total >= 100 :
```

If statements allow you to specify code that only executes if a specific condition is true

```
answer=input("Would you like express shipping? ")
if answer == "yes" :
    print("That will be an extra $10")
print("Have a nice day")
```

Does it matter if that print statement is indented?

YES – the indented code is only executed if the condition is true

Almost every if statement can be written two ways

```
if answer == "yes" :
if not answer == "no" :

if total < 100 :
if not total >= 100 :
```

Which do you prefer?

What do you think will happen if we type "YES" instead of "yes"

```
answer=input("Would you like express shipping? ")
if answer == "yes" :
     print("That will be an extra $10")
print("Have a nice day")
```

One of the challenges of working with strings of characters, is that the computer considers "y" and "Y" to be two different letters.

Is there a way we could change a string from uppercase to lowercase?

```
answer=input("Would you like express shipping?")
if answer.lower() == "yes" :
     print("That will be an extra $10")
print("Have a nice day")
```

Hint: There were functions we could call for variables

Hint: lower()

What if we try an if statement with numbers instead of strings

```
deposit = 150
if deposit > 100 :
     print("You get a free toaster!")
print("Have a nice day")
```

What will appear on the screen if deposit is 150?

What will appear on the screen if deposit is 50?

What will appear on the screen if deposit is 100?

Always test should it be >,< or <=, >=

```
deposit = 150
if deposit > 100 :
     print("You get a free toaster!")
print("Have a nice day")
```

What will appear on the screen if deposit is 150?

What will appear on the screen if deposit is 50?

What will appear on the screen if deposit is exactly 100?

How could we let the user enter the amount to deposit?

```
deposit=input("How much would you like to deposit? ")
if deposit > 100 :
     print("You get a free toaster!")
print("Have a nice day")
```

Why did our code crash?

How can we fix it?

We have to convert the string value returned by the input function to a number

```
deposit=input("How much would you like to deposit? ")
if float(deposit) > 100 :
    print("You get a free toaster!")
print("Have a nice day")
```

What if you get a free toaster for over $100 and a free mug for under $100

```
deposit=input("How much would you like to deposit? ")
if float(deposit) > 100 :
        print("You get a free toaster!")
else:
        print("Enjoy your mug!")
print("Have a nice day")
```

The code in the ***else*** statement is only executed if the condition is NOT true

What will appear on the screen if we enter 50? 150? 100?

What if you get a free tv for over $1000, a toaster for over $100 and a free mug for under $100!

```
deposit=input("How much would you like to deposit? ")
if float(deposit) > 100 :
        print("You get a free toaster!")
else:
        print("Enjoy your mug!")
print("Have a nice day")
```

The code in the ***else*** statement is only executed if the condition is NOT true

What will appear on the screen if we enter 50? 150? 100?

You can use boolean variables to remember if a condition is true or false

```
deposit= input("how much would you like to deposit? ")
if float(deposit) > 100 :
     #Set the boolean variable freeToaster to True
     freeToaster=True

#if the variable freeToaster is True
#the print statement will execute
if freeToaster :
     print("enjoy your toaster")
```

Make sure you test what happens when your if statement is true and what happens when your if statement is false.

Why does our code crash when we enter a value of 50 for a deposit?

```
deposit= input("how much would you like to deposit? ")
if float(deposit) > 100 :
    #Set the boolean variable freeToaster to True
    freeToaster=True

#if the variable freeToaster is True
#the print statement will execute
if freeToaster :
    print("enjoy your toaster")
```

Look at the error message: **Name ‘freeToaster’ is not defined.**

It's always a good idea to initialize your variables!

```
#Initialize the variable to fix the error
freeToaster=False

deposit= input("how much would you like to deposit? ")
if float(deposit) > 100 :
     #Set the boolean variable freeToaster to True
     freeToaster=True

#if the variable freeToaster is True
#the print statement will execute
if freeToaster :
     print("enjoy your toaster")
```

Aren’t you just making the code more complicated by using the Boolean variable?

- That depends…
- What if you are writing a program, and there is more than one place you have to check that condition? You could check the condition once and remember the result in the Boolean variable
- What if the condition is very complicated to figure out? It might be easier to read your code if you just use a Boolean variable (often called a flag) in your if statement

And now we have more ways to make typing mistakes! Can you find three?

```
deposit=input("How much would you like to deposit? ")
if float(deposit) > 100
      print("You get a free toaster!")
freeToaster=true
else:
       print("Enjoy your mug!")
print("Have a nice day")
```

```
deposit=input("How much would you like to deposit? ")
if float(deposit) > 100 :
      print("You get a free toaster!")
      freeToaster=True
else:
       print("Enjoy your mug!")
print("Have a nice day")
```

MVA Microsoft Virtual Academy

Handling more complex conditions

Microsoft

And/or, nested if, switch

Sometimes you have multiple conditions that affect what you want to happen

- If you are in Canada say hello in English, if you are in Germany use German, if you are in France use French, …
- If you win the lottery and the prize is over a million dollars then retire to a life of luxury
- If it is Monday, check to see if there is fresh coffee. If there is no fresh coffee go to the nearest café

If you are in Canada say hello in English, if you are in Germany use German, if you are in France use French, …

- This is an interesting situation because you really only have one condition to check, but that one condition could have many different values

You can use an “elif” to check for different values

```
country = input("Where are you from? " )

if country == "CANADA" :
     print("Hello")
elif country == "GERMANY" :
     print("Guten Tag")
elif country == "FRANCE" :
     print("Bonjour")
```

Note that the elif statement is not indented!

“elif” is short for Else if

What if someone enters a country we didn’t list?

We should add an “**else**” statement to catch any conditions we didn’t list

```
country = input("Where are you from? " )

if country == "CANADA" :
     print("Hello")
elif country == "GERMANY" :
     print("Guten Tag")
elif country == "FRANCE" :
     print("Bonjour")
else :
     print("Aloha/Ciao/G’Day")
```

If you win the lottery and the prize is over a million dollars then retire to a life of luxury

- Sometimes the decision on whether to take the next step depends on a combination of factors
- If I win the lottery, but only win $5 I can’t retire
- If the lottery gives out a million dollars but I didn’t win, I can’t retire
- I can only retire if I win the lottery and the prize was over a million dollars

Using “and” allows you to require multiple conditions to be true

The “and” is only evaluated as True if both conditions are True.

```
#Imagine you have code that ran earlier which
#set these two variables
wonLottery = True
bigWin = True

#print statement only executes if both conditions are true
if wonLottery and bigWin :
     print("you can retire")
```

Here are all the possible combinations

```
if firstCondition and secondCondition :
```

First Condition is	Second Condition is	Statement is
True	True	True
True	False	False
False	True	False
False	False	False

Sometimes we want to do something if one condition "or" another is True

- If it's Saturday or Sunday I can sleep in
- If it's raining or snowing don't bike to work

Using “or” allows you to require only one of two or more conditions to be true

The “or” is evaluated as True if either of the conditions is True.

```
#Imagine you have code that ran earlier which
#set these two variables
saturday = True
sunday = False

#print statement executes if either condition is true
if saturday or sunday :
     print("you can sleep in")
```

Here are all the possible combinations

```
if firstCondition or secondCondition :
```

First Condition is	Second Condition is	Statement is
True	True	True
True	False	True
False	True	True
False	False	False

You can combine multiple “and”/“or” in a single if statement

```
if month == "Sep" or month =="Apr"
or month == "Jun" or month == "Nov" :
     print("There are 30 days in this month")

if favMovie == "Star Wars"
and favBook == "Lord of the Rings"
and favEvent == "ComiCon" :
     print("You and I should hang out")
```

You can combine “and”/”or” in a single statement

```
if country == "CANADA" and
pet == "MOOSE" or  pet == "BEAVER" :
     print("Do you play hockey too?")
```

Make sure you test different combinations

- Country = CANADA, Pet = MOOSE
- Country = CANADA, Pet = BEAVER
- Country = VIETNAM, Pet = MOOSE
- Country = VIETNAM, Pet = BEAVER

This one doesn’t seem to work the way you would expect!

Do you remember learning order of operations for math in school?

- 8+5*2=?
- Multiplication and Division are done before addition and subtraction
- 8+5*2 = 18

There is an order of operations for “and”/”or”: “and” are evaluated first

```
if country == "CANADA" and pet == "MOOSE"
or  pet == "BEAVER" :
     print("Do you play hockey too?")
```

In math, how can you specify that you want to do addition before multiplication?

- Use parentheses!
- (8+5)*2 = 26

We can use parentheses to execute “or” before “and”

```
if country == "CANADA" and
(pet == "MOOSE" or  pet == "BEAVER") :
     print("Do you play hockey too")
```

When in doubt, just add parentheses whenever you combine and/or in a single if statement.

It might be redundant, but it will be easier for someone to read your code and you are less likely to make mistakes.

Sometimes we have multiple conditions but just using and "and"/"or" may not work

- How could you handle this in code?
- If it is Monday, go check to see if there is fresh coffee. If there is no fresh coffee go to the nearest café
- In this situation you have to check a condition, if it is true you want to check another condition.

You can nest if statements inside each other

```
monday = True
freshCoffee = False
if monday :
    #you could have code here to check for fresh coffee

    # the if statement is nested, so this if statement
    # is only executed if the other if statement is true
    if not freshCoffee :
        print("go buy a coffee!")
    print("I hate Mondays")
print("now you can start work")
```

You have to be VERY careful with how the code is indented, because that determines which code goes with which if statement

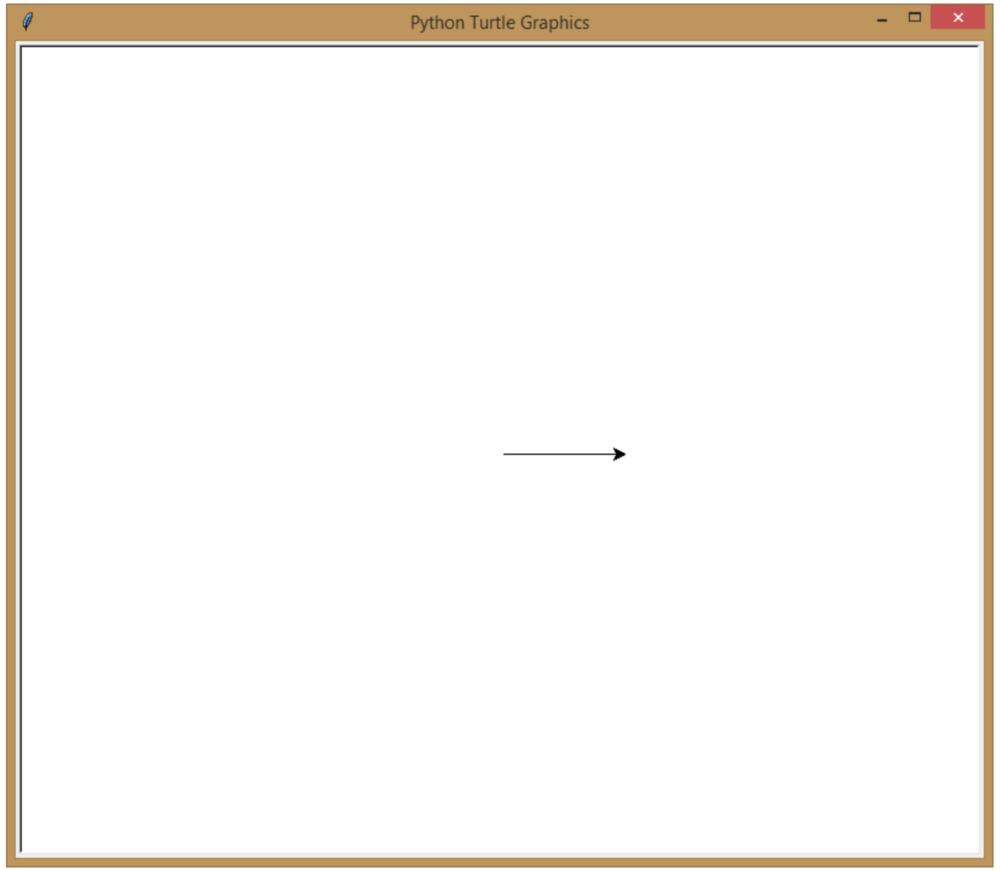
Repeating yourself

Repeating code

For loops

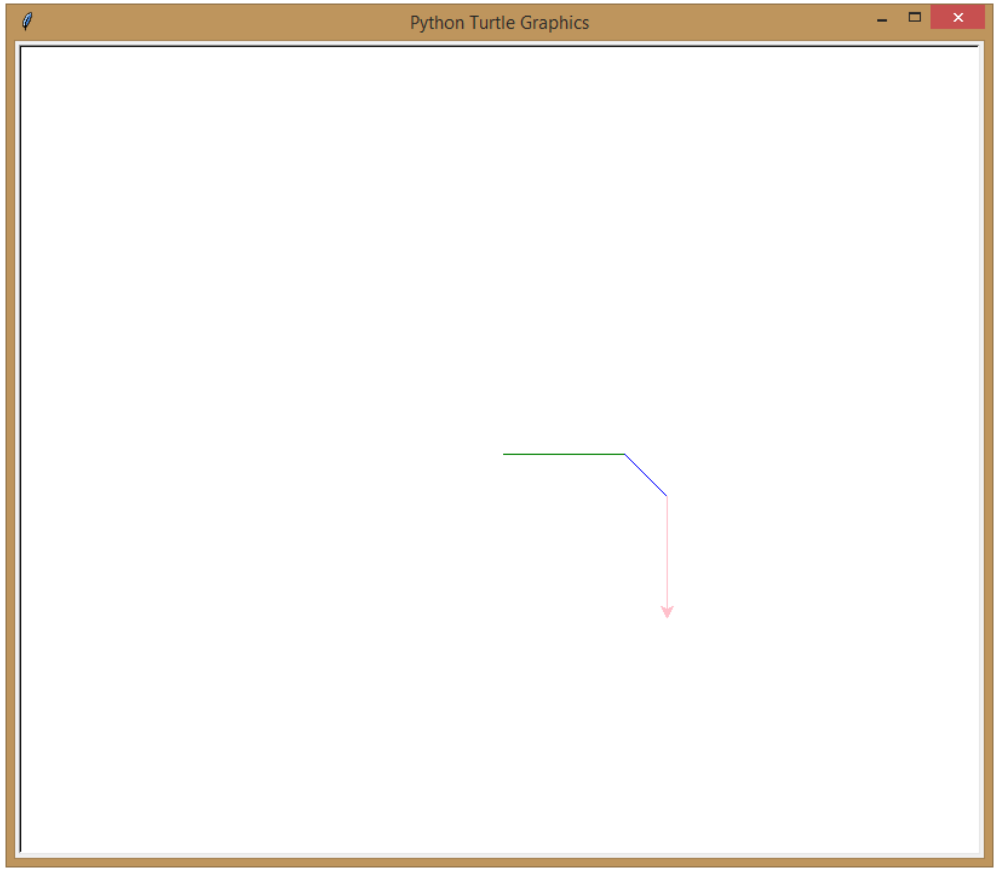
Did you know Python works as an Etch a Sketch?

```
import turtle
turtle.forward(100)
```



turtle likes to draw ☺

```
import turtle
turtle.color('green')
turtle.forward(100)
turtle.right(45)
turtle.color('blue')
turtle.forward(50)
turtle.right(45)
turtle.color('pink')
turtle.forward(100)
```

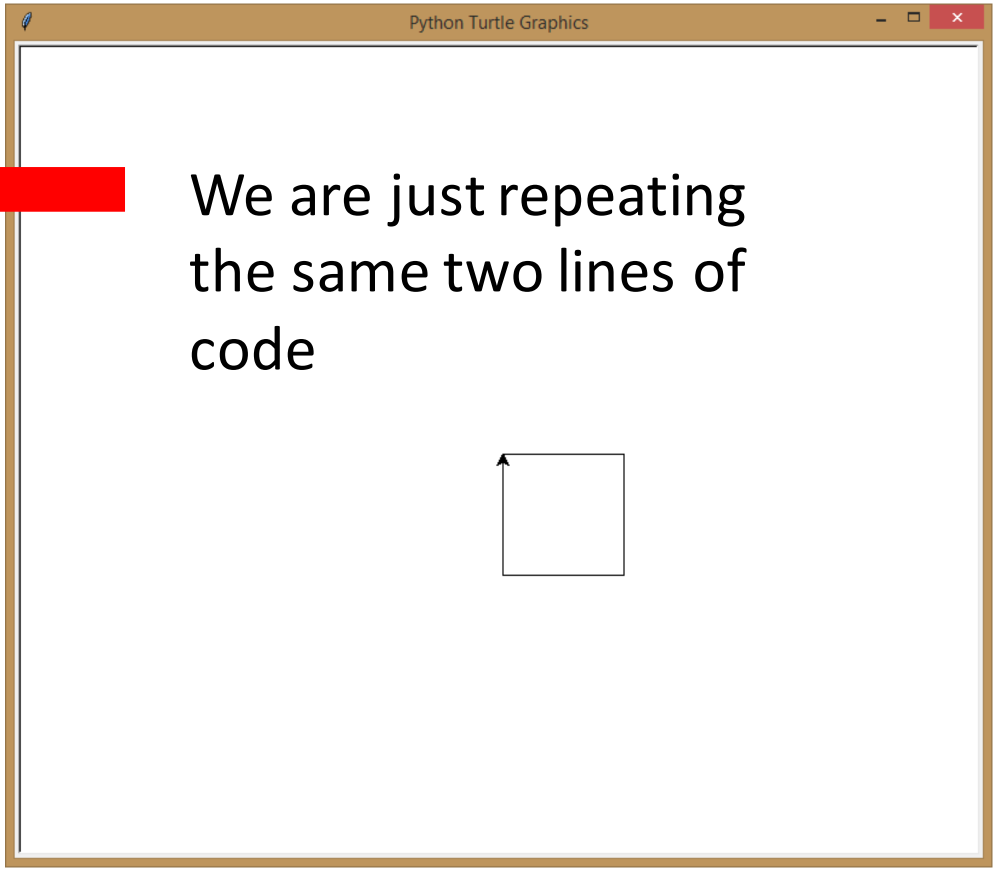


You can probably guess what some of the turtle commands do

`right(x)`	Rotate right x degrees
`left(x)`	Rotate left x degrees
`color(‘x’)`	Change pen color to x
`forward(x)`	Move forward x
`backward(x)`	Move backward x

How would we get turtle do draw a square?

```
import turtle
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
```



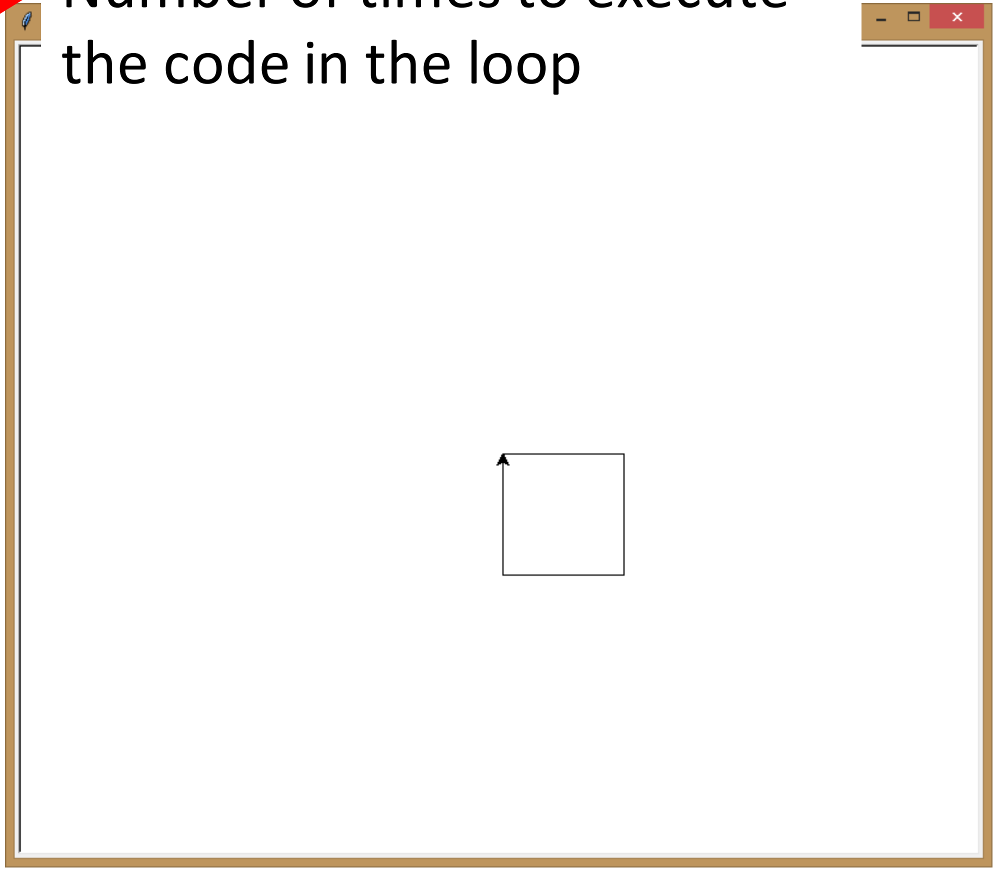
We are just repeating the same two lines of code

Loops allow us to repeat the same line of code as often as we want

```
import turtle
for steps in range(4):
        turtle.forward(100)
        turtle.right(90)
```

Number of times to execute the code in the loop

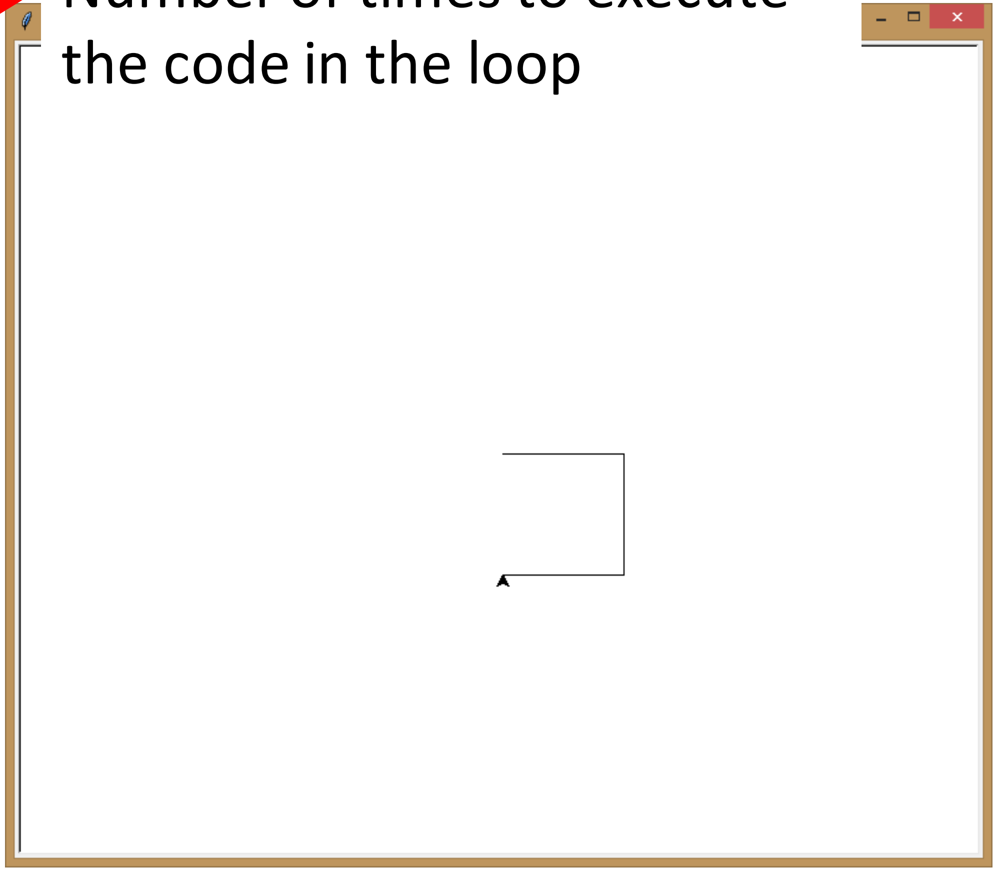
You MUST indent the code you want repeated



When you change the range, you change the number of times the code executes

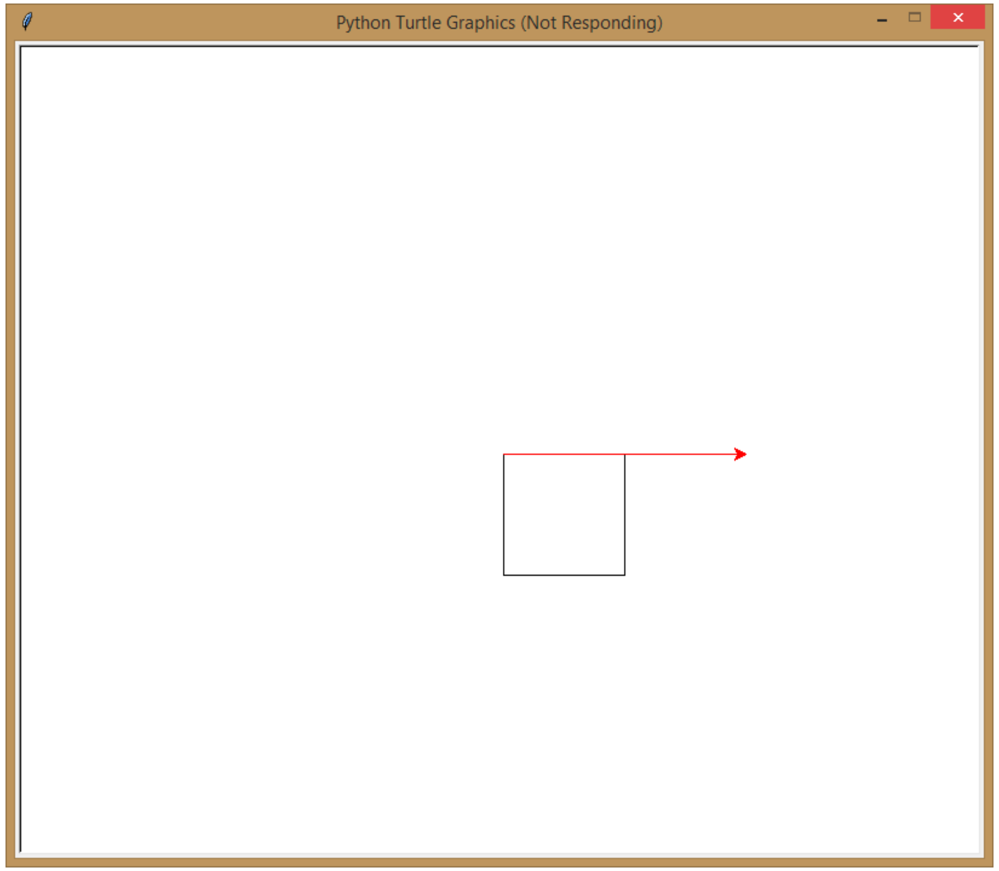
```
import turtle
for steps in range(3):
    turtle.forward(100)
    turtle.right(90)
```

Number of times to execute the code in the loop



Only the indented code is repeated!

```
import turtle
for steps in range(4):
    turtle.forward(100)
    turtle.right(90)
turtle.color('red')
turtle.forward(200)
```



Now we have new ways to mess up our code!

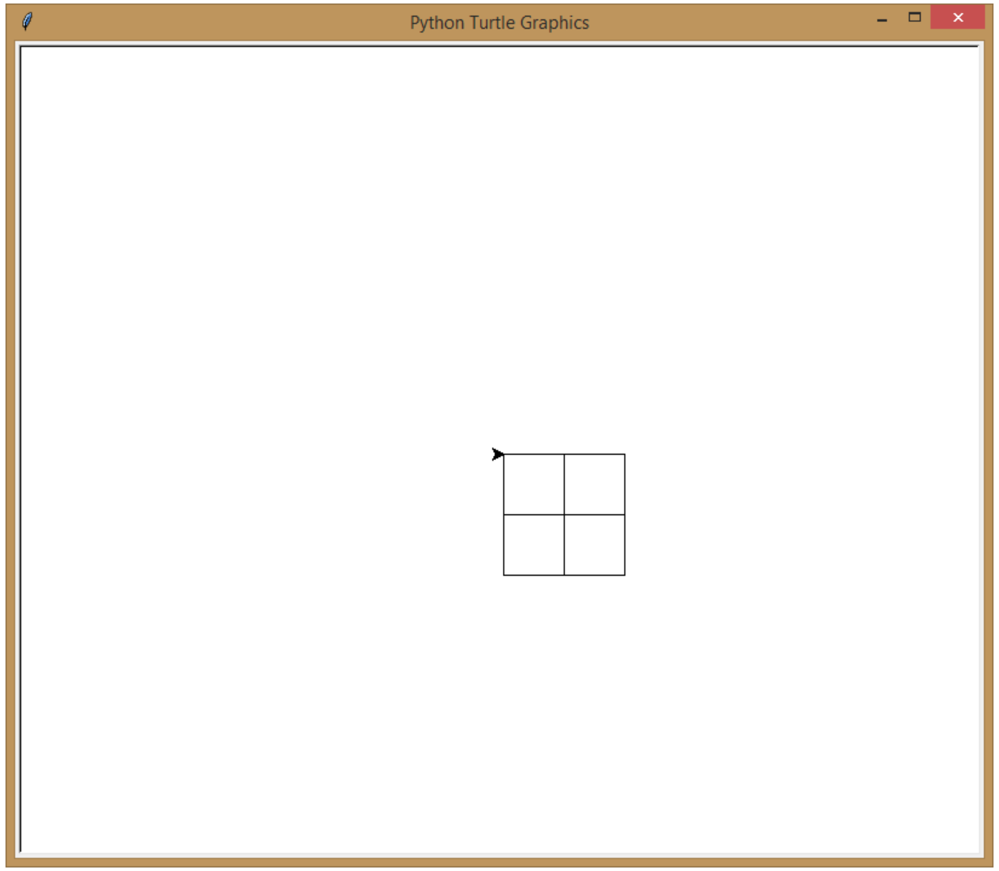
Can you find three mistakes in this code?

```
improt turtle
for steps in range(4)
    turtle.forward(100)
turtle.right(90)
```

```
import turtle
for steps in range(4):
    turtle.forward(100)
    turtle.right(90)
```

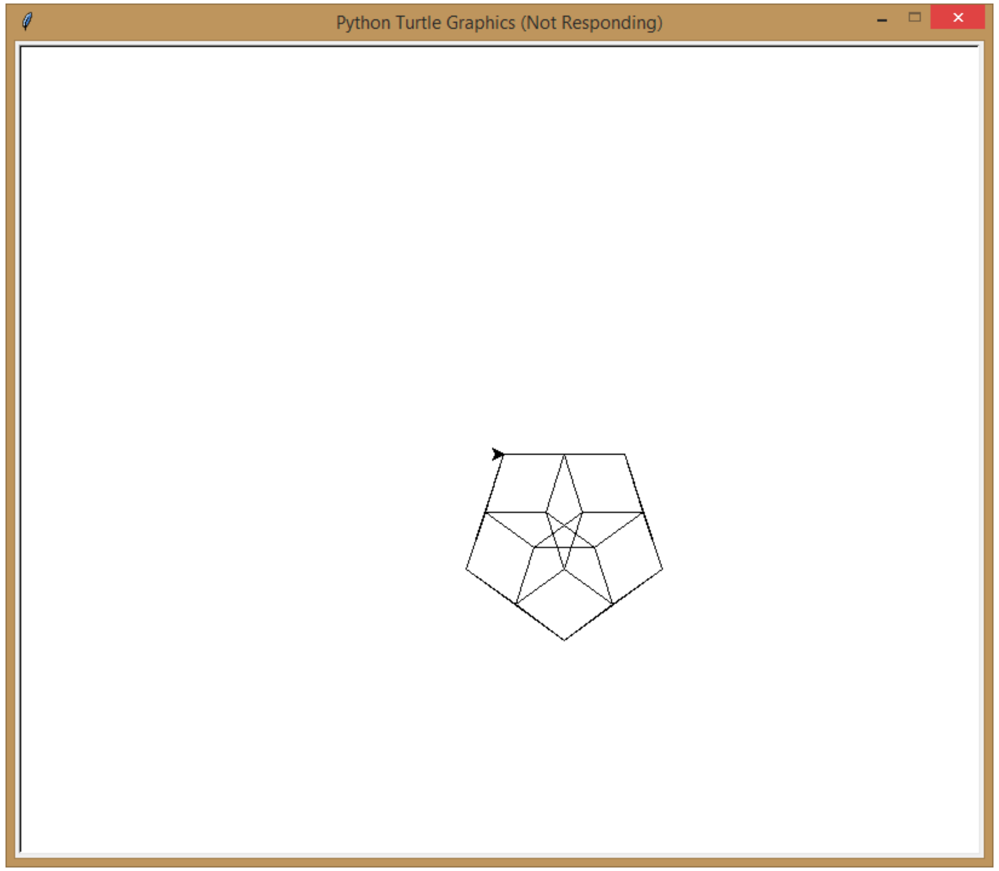
You can have lots of fun when you put a loop inside another loop!

```
import turtle
for steps in range(4):
     turtle.forward(100)
     turtle.right(90)
     for moresteps in range(4):
          turtle.forward(50)
          turtle.right(90)
```



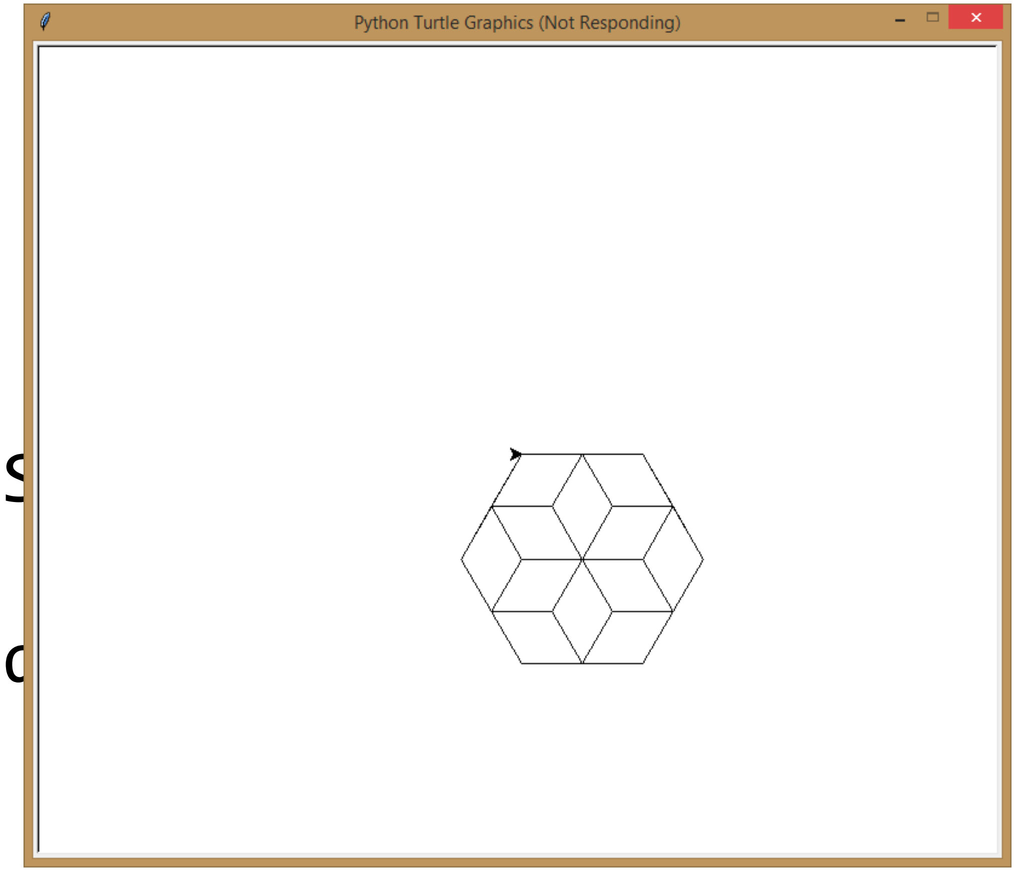
Just for fun

```
import turtle
for steps in range(5):
    turtle.forward(100)
    turtle.right(360/5)
    for moresteps in range(5):
        turtle.forward(50)
        turtle.right(360/5)
```



We could use a variable to decide the number of sides our object will have

```
import turtle
nbrSides = 6
for steps in range(nbrSides):
        turtle.forward(100)
        turtle.right(360/nbrSides)
        for moresteps in range(nbrS
              turtle.forward(50)
              turtle.right(360/nbrSic
```



What's the advantage of using a variable here instead of just typing in the number?

```
import turtle
nbrSides = 6
for steps in range(nbrSides):
     turtle.forward(100)
     turtle.right(360/nbrSides)
     for moresteps in range(nbrSides):
         turtle.forward(50)
         turtle.right(360/nbrSides)
```

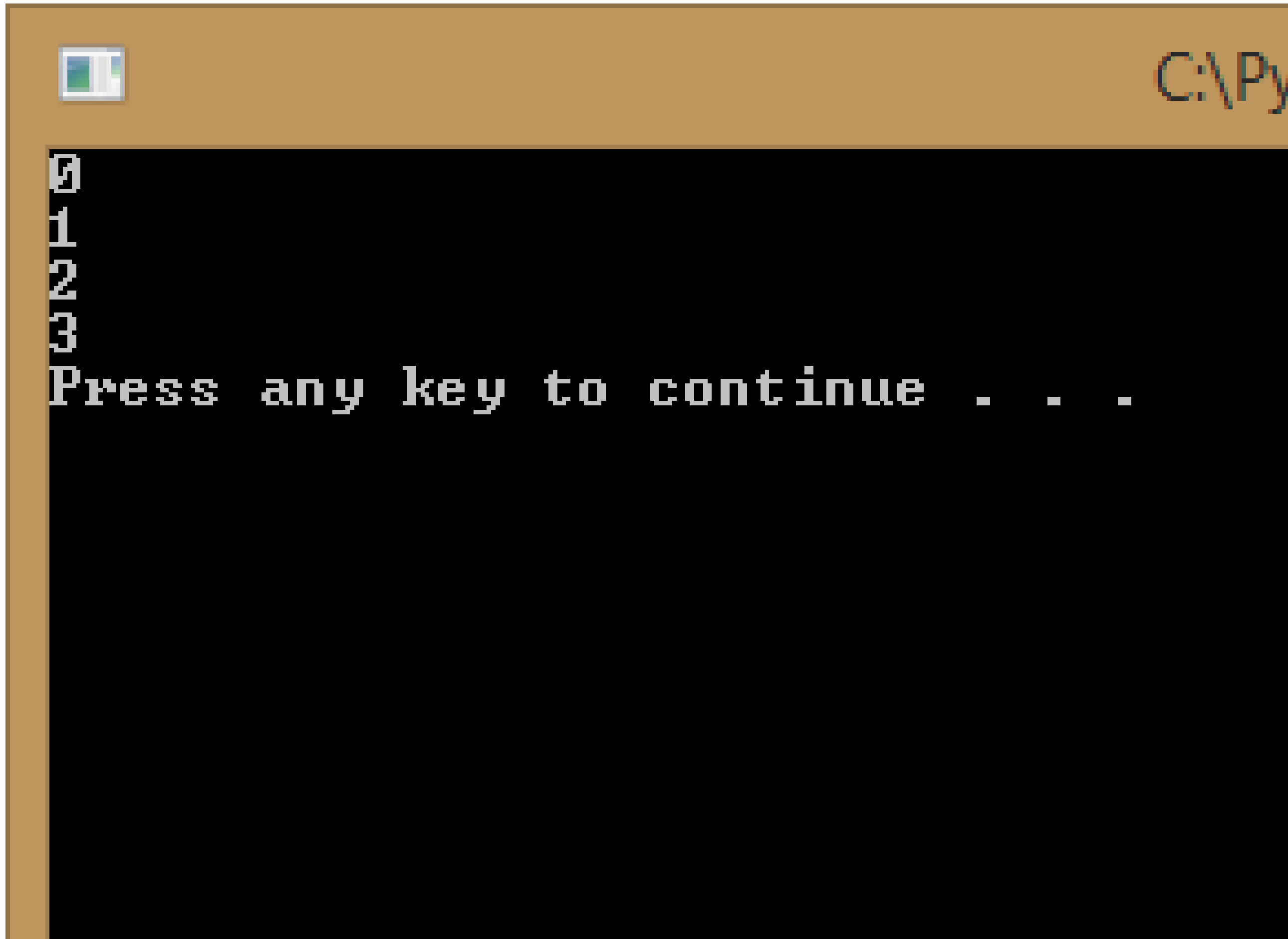
When we use a variable and we want to change a value that appears in many places, we only have to update one line of code!

```
import turtle
nbrSides = 6
for steps in range(nbrSides):
        turtle.forward(100)
        turtle.right(360/nbrSides)
        for moresteps in range(nbrSides):
              turtle.forward(50)
              turtle.right(360/nbrSides)
```

Did you know you can actually look at the values being used in the loop?

```
for steps in range(4) :
    print(steps)
```

Yes, counting starts at zero in for loops, that's pretty common in programming

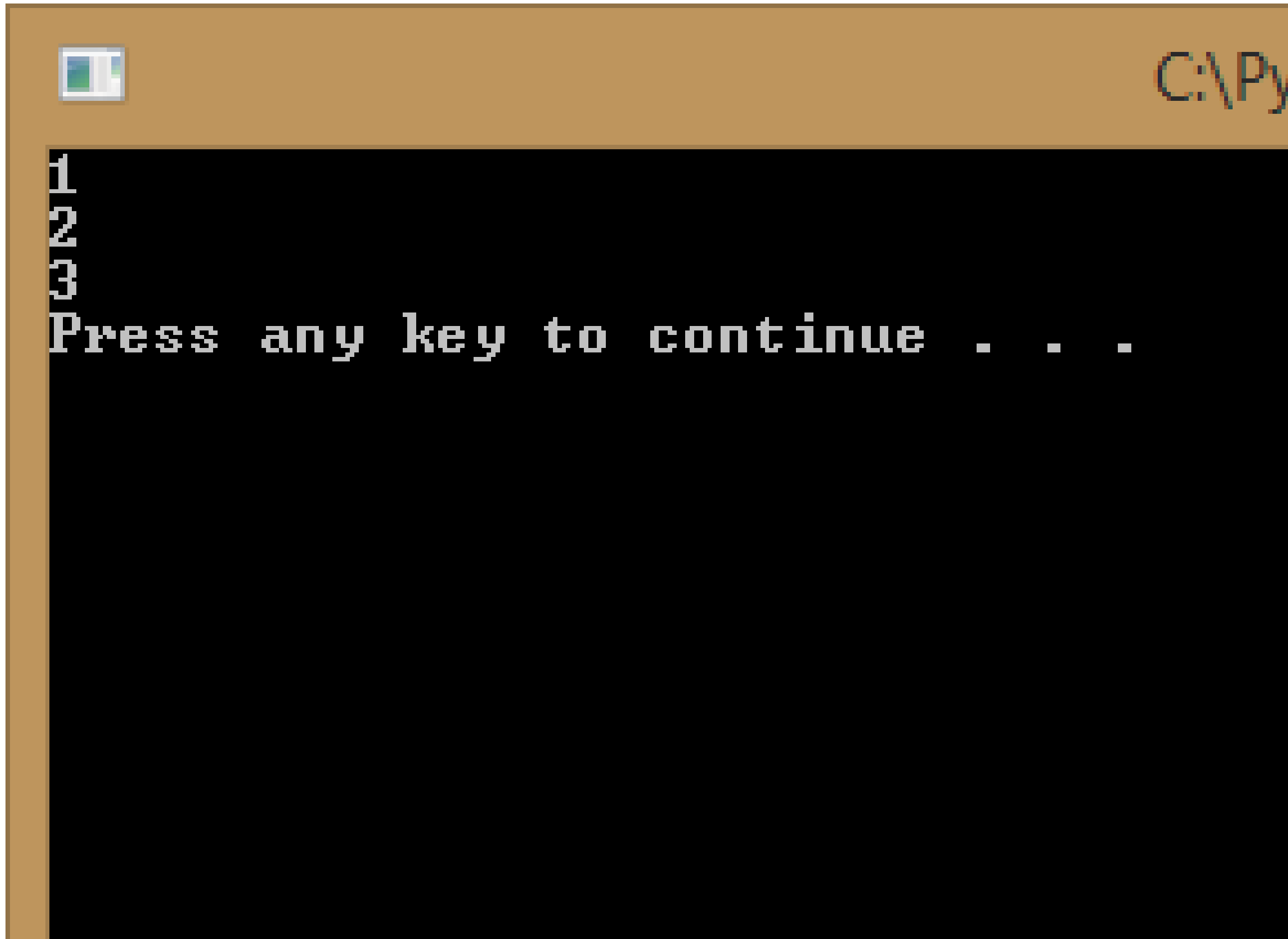


```
0
1
2
3
Press any key to continue . . .
```

If you need to start counting from "1" you can specify numbers to count to and from

```
for steps in range(1,4) :
    print(steps)
```

Did you notice this time the loop only executed three times?

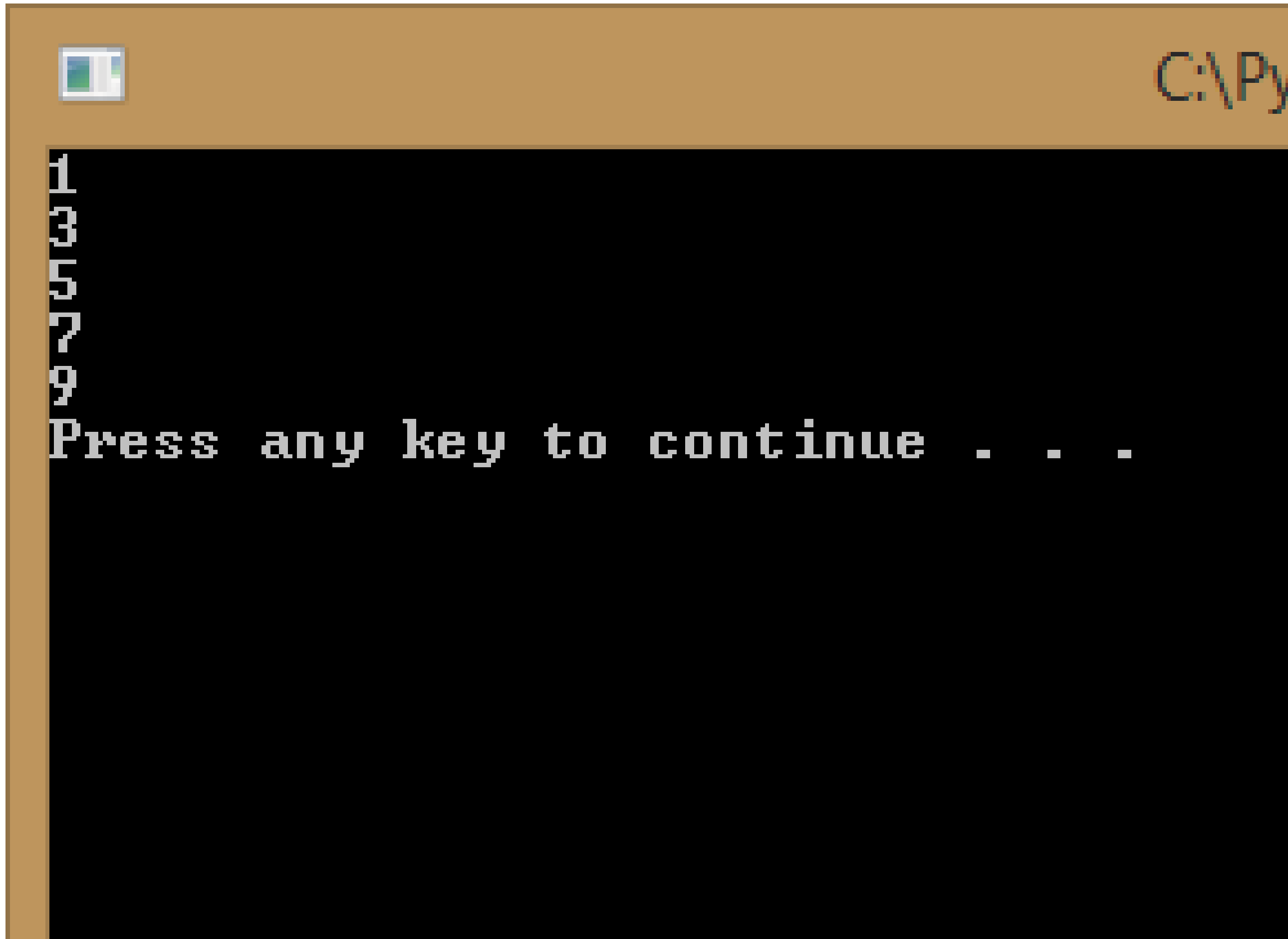


C:\Py

```
1
2
3
Press any key to continue . . .
```

You can also tell the loop to skip values by specifying a step

```
for steps in range(1,10,2) :
    print(steps)
```



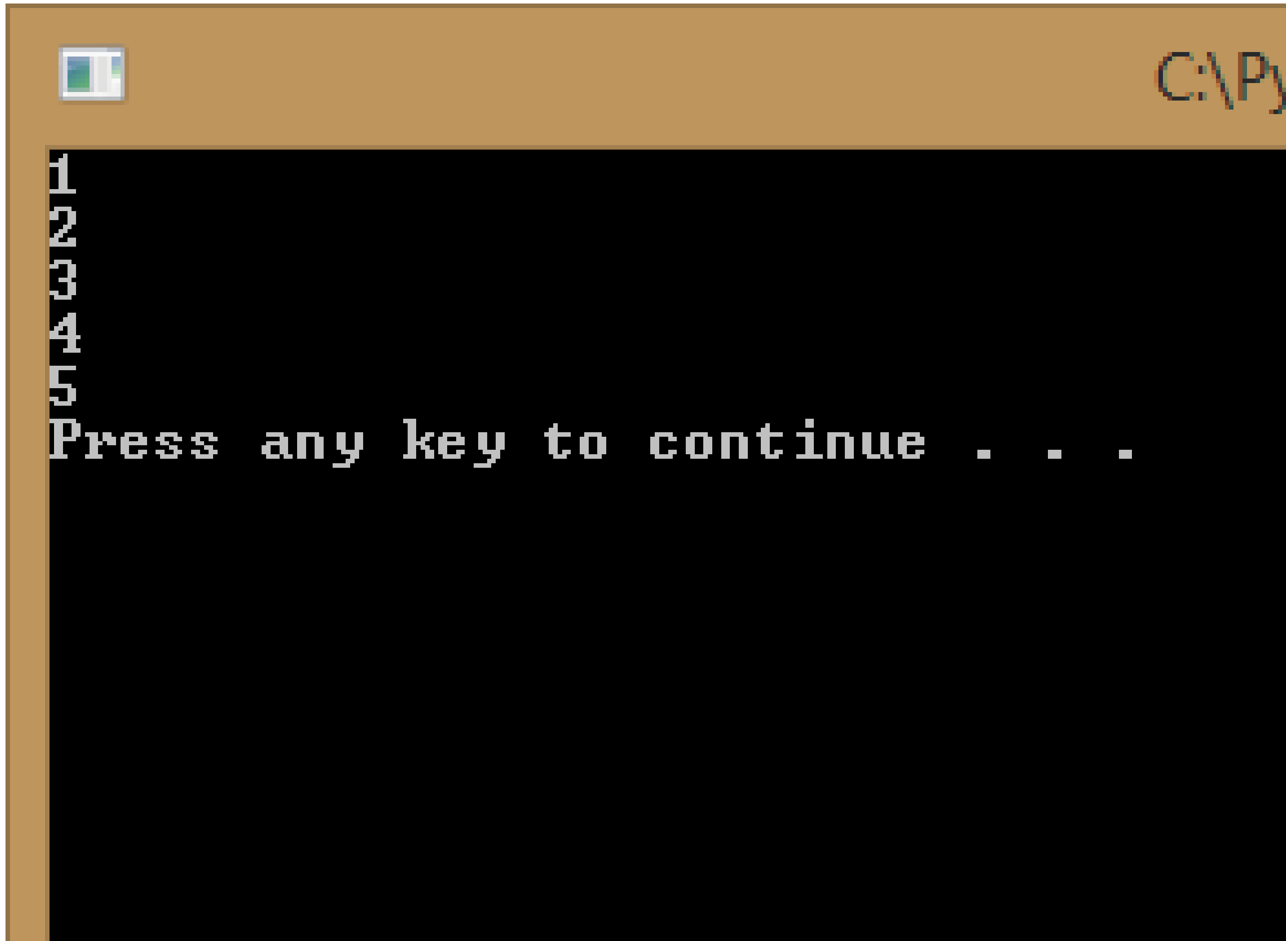
C:\Py

```
1
3
5
7
9
Press any key to continue . . .
```

One of the cool things about Python is the way you can tell it exactly what values you want to use in the loop

```
for steps in [1,2,3,4,5]:
        print(steps)
```

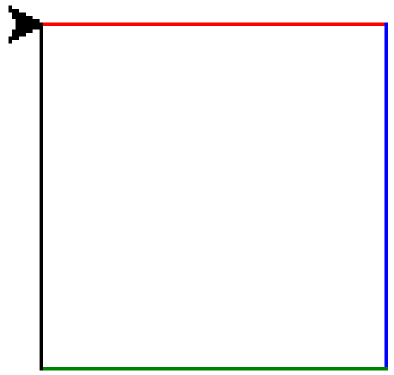
This does require [] brackets instead of () and you don't use the "range" keyword



```
1
2
3
4
5
Press any key to continue . . .
```

And you don’t have to use numbers!

```
import turtle
for steps in ['red','blue','green','black'] :
    turtle.color(steps)
    turtle.forward(100)
    turtle.right(90)
```

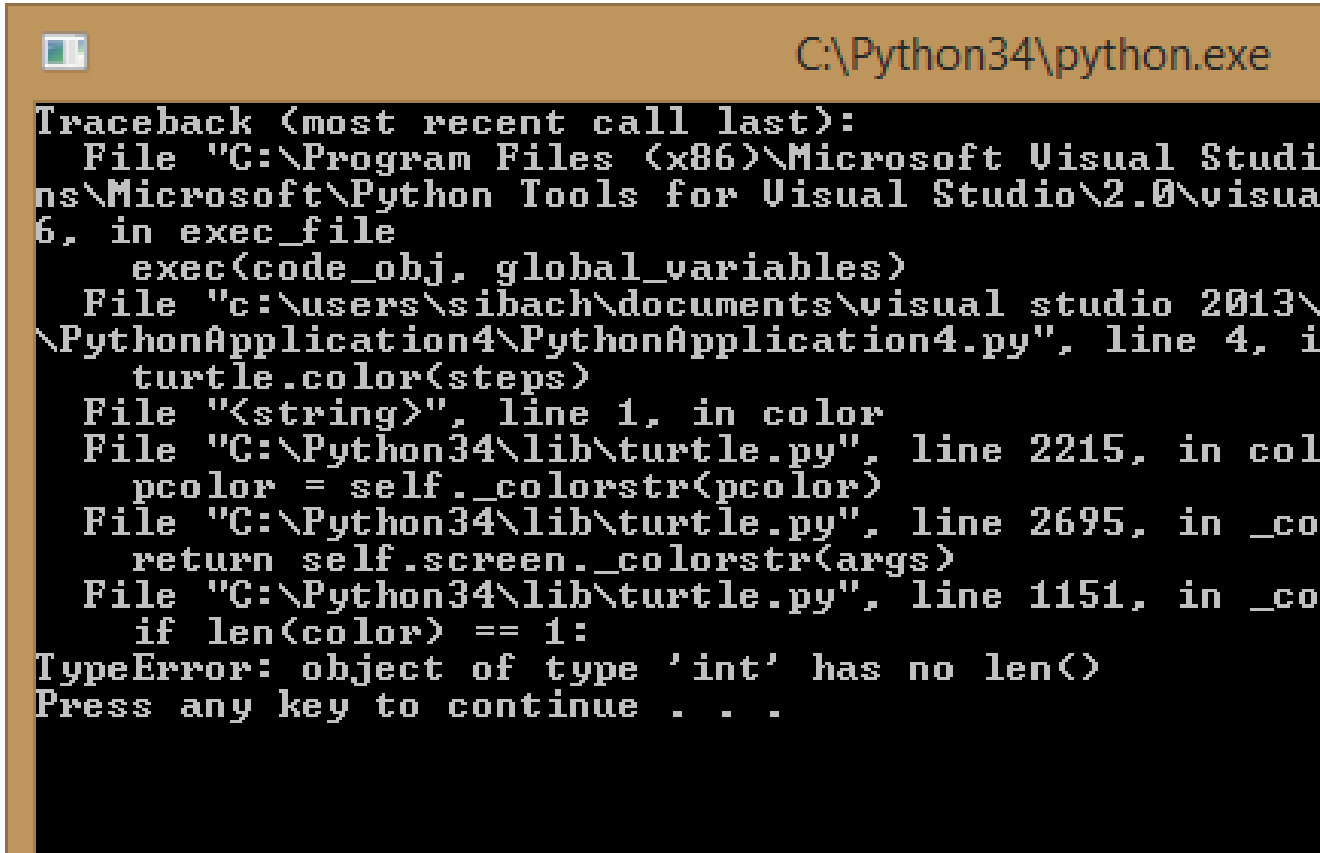


What do you think this code will do?

You can even mix up different datatypes (e.g. numbers and strings) but…

```
import turtle
for steps in ['red','blue','green','black',8] :
        turtle.color(steps)
        turtle.forward(100)
        turtle.right(90)
```

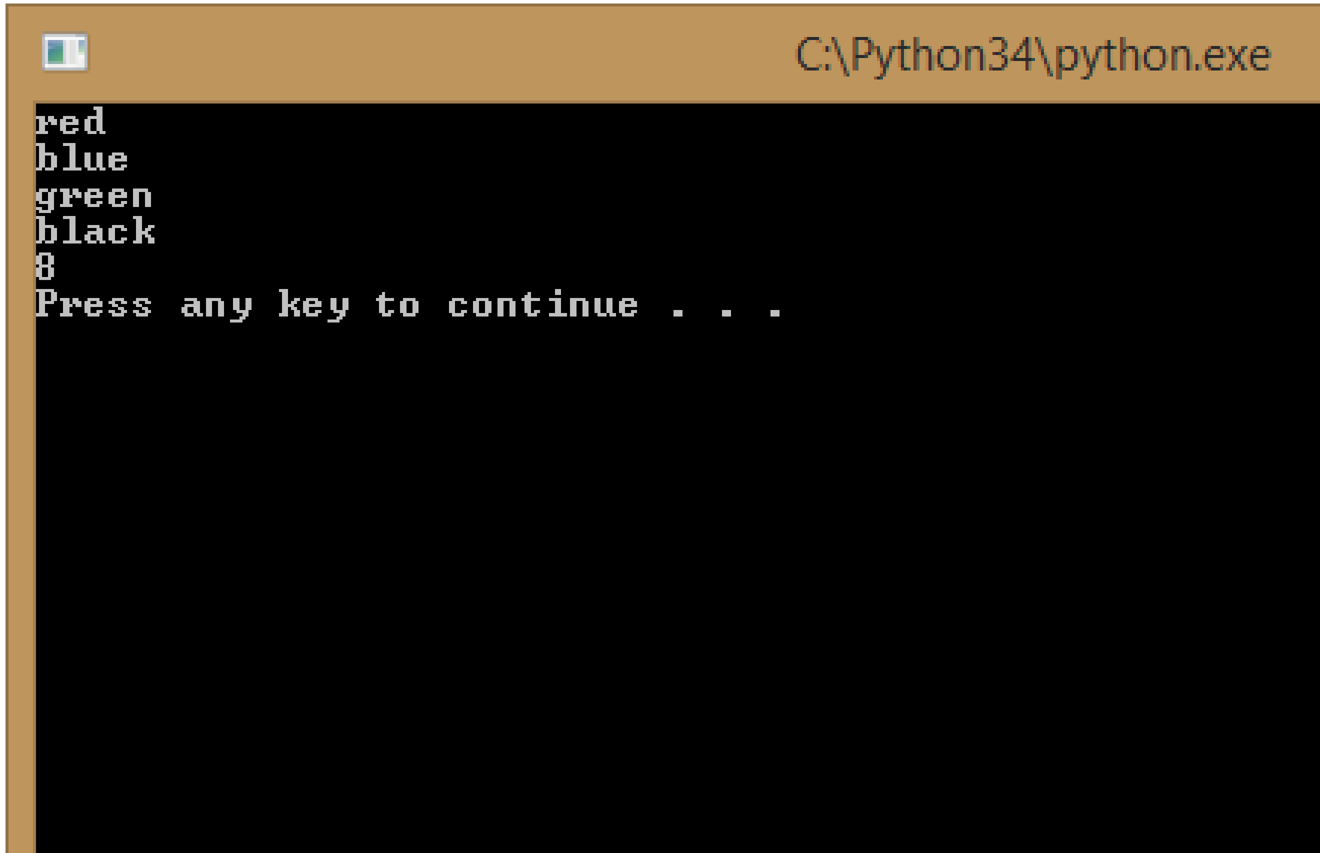
You had better make sure any code using that value can handle the datatype!


C:\Python34\python.exe

```
Traceback (most recent call last):
  File "C:\Program Files (x86)\Microsoft Visual Studi
ns\Microsoft\Python Tools for Visual Studio\2.0\visua
6, in exec_file
    exec(code_obj, global_variables)
  File "c:\users\sibach\documents\visual studio 2013\
\PythonApplication4\PythonApplication4.py", line 4, i
    turtle.color(steps)
  File "<string>", line 1, in color
  File "C:\Python34\lib\turtle.py", line 2215, in col
    pcolor = self._colorstr(pcolor)
  File "C:\Python34\lib\turtle.py", line 2695, in _co
    return self.screen._colorstr(args)
  File "C:\Python34\lib\turtle.py", line 1151, in _co
    if len(color) == 1:
TypeError: object of type 'int' has no len()
Press any key to continue . . .
```

You can’t set the color to a number so the code crashed, but the print can accept strings or numbers

```
for steps in ['red','blue','green','black‘,8] :
    print (steps)
```



```
red
blue
green
black
8
Press any key to continue . . .
```

While Loops

While loops

For loops allow us to execute code a fixed number of times

```
import turtle
for steps in range(4):
    turtle.forward(100)
    turtle.right(90)
```

What if we don’t know how exactly how many times to repeat the code?

- Ask for the names of everyone in a class and print each name
- Keep guessing until you get the right answer
- Read all the values in a file or in a database

While Loops allow you to execute until a particular condition is true

```
answer = "0"

while answer != "4":
     answer = input("What is 2 + 2 ")

print ("Yes! 2 + 2 = 4")
```

You have to declare the variable before you use it in the loop

Execute the code in the loop over and over while the variable answer is not equal to 4

Remember only the indented code is repeated, so this command only executes after you exit the loop

Can you figure out what this code will do?

```
import turtle
counter = 0
while counter < 4:
     turtle.forward(100)
     turtle.right(90)
     counter = counter+1
```

Yes, it will draw a square

While loops can be used instead of for loops

```
import turtle
for steps in range(4):
    turtle.forward(100)
    turtle.right(90)
```

```
import turtle
counter = 0
while counter < 4:
    turtle.forward(100)
    turtle.right(90)
    counter = counter+1
```

Both loops have the same end result

Can you figure out what this code will do?

```python
import turtle
counter = 0
while counter <= 4:
     turtle.forward(100)
     turtle.right(90)
     counter = counter+1
```

It will actually draw 5 lines! Not 4!

Can you figure out what this code will do?

```python
import turtle
counter = 1
while counter < 4:
     turtle.forward(100)
     turtle.right(90)
     counter = counter+1
```

It will draw only 3 lines! Not 4!

How often will this loop execute?

```
import turtle
counter = 0
while counter < 3:
     turtle.forward(100)
     turtle.right(90)
```

Trick question! It will execute forever! Because the value of counter is never updated! How can counter ever become greater than 3? This is called an endless loop. Yet another way to mess up our code

It’s easier to make a mistake with a while loop than a for loop

- Use for loops whenever possible

Don’t fear the while loop

- There is a time and place to use while loops, in particular they are really useful when you want to read data until there is no more data to read.

MVA Microsoft Virtual Academy

Arrays and lists

Microsoft

Sometimes you need to store more than one value

- I want to remember the names of everyone coming to a party
- I want to remember the scores I got in all my courses

Lists allow you to store multiple values

```python
guests = ['Christopher','Susan','Bill','Satya']

scores = [78,85,62,49,98]
```

You can create an empty list and add values later

```
guests = []

scores = []
```

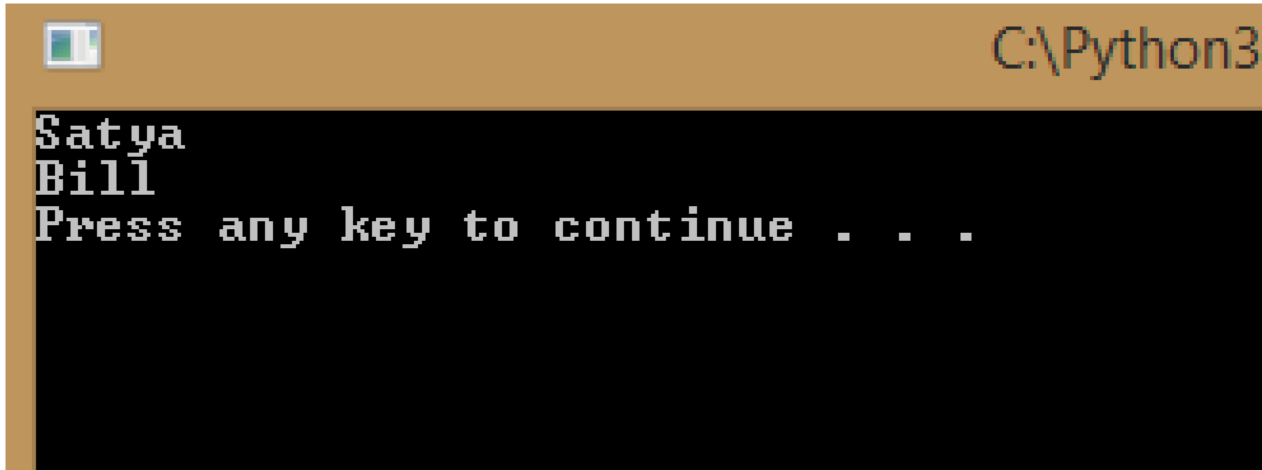
You can reference any value in the list by specifying it's position in the list

```
guests = ['Christopher','Susan','Bill','Satya']

#print the first guest
#the first value is in position 0
print(guests[0])

scores = [78,85,62,49,98]
#Print the fourth score
print(scores[3])
```

geek tip: We call the position in the list the **index**

You can count backwards to specify the entry you want

```
guests = ['Christopher','Susan','Bill','Satya']

#print the last entry in the list
print(guests[-1])

#print the second last entry in the list
print(guests[-2])
```

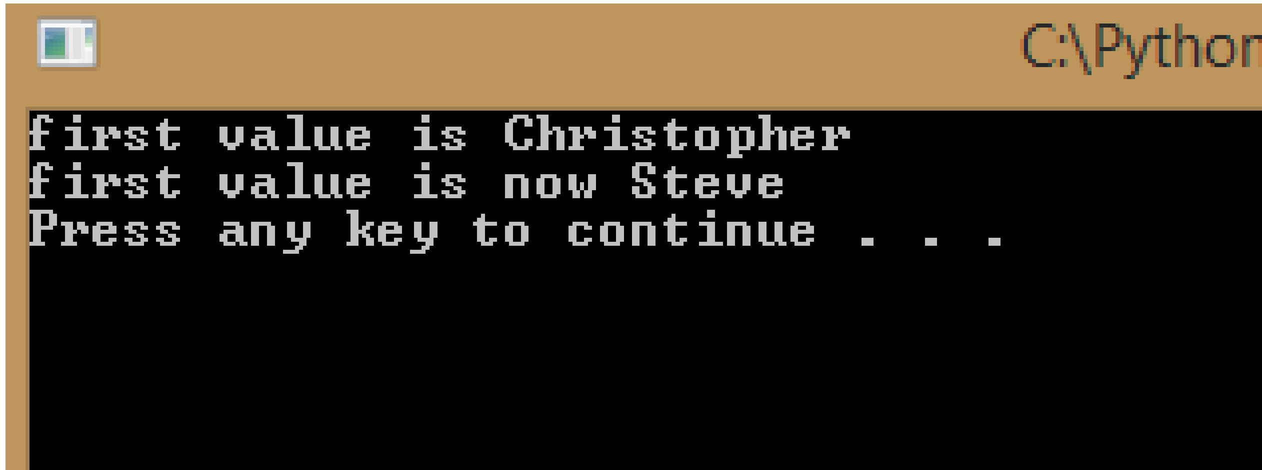


```
Satya
Bill
Press any key to continue . . .
```

What if you want to change the list contents?

You can change a value in a list

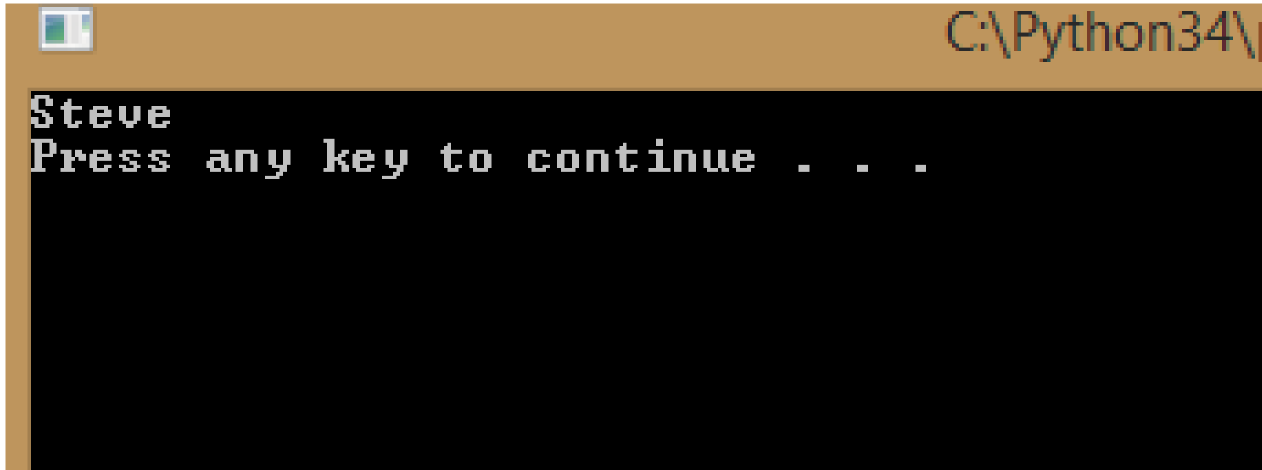
```
guests = ['Christopher','Susan','Bill','Satya']
print("first value is " + guests[0])

#change the first value in the list to Steve
guests[0] = 'Steve'
print("first value is now " + guests[0])
```



```
first value is Christopher
first value is now Steve
Press any key to continue . . .
```

You can add a value to a list with append()

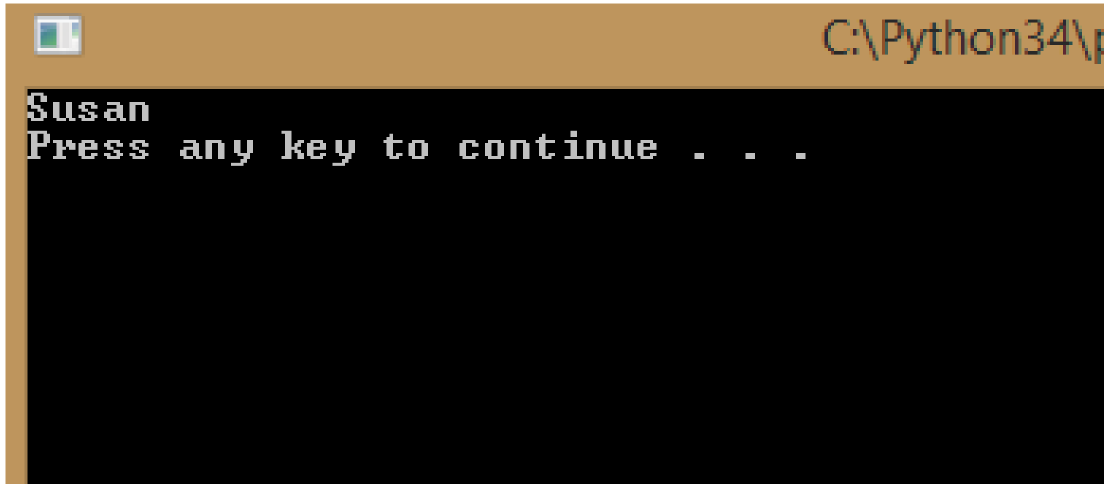
```
guests = ['Christopher','Susan','Bill','Satya']

#add a new value to the end of the list
guests.append('Steve')

#display the last value in the list
print(guests[-1])
```



```
Steve
Press any key to continue . . .
```

You can remove a value from a list with remove()

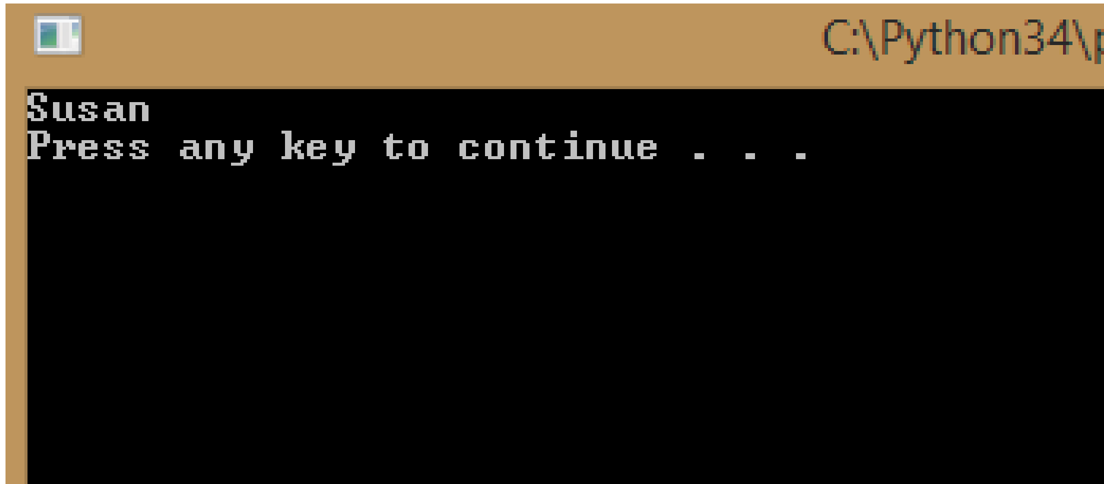
```
guests = ['Christopher','Susan','Bill','Satya']

#add a new value to the end of the list
guests.remove('Christopher')

#display the last value in the list
print(guests[0])
```



```
Susan
Press any key to continue . . .
```

You can use the del command to delete an entry

```
guests = ['Christopher','Susan','Bill','Satya']

#delete the first item in the list
del guests[0]

#print the first item in the list
print(guests[0])
```

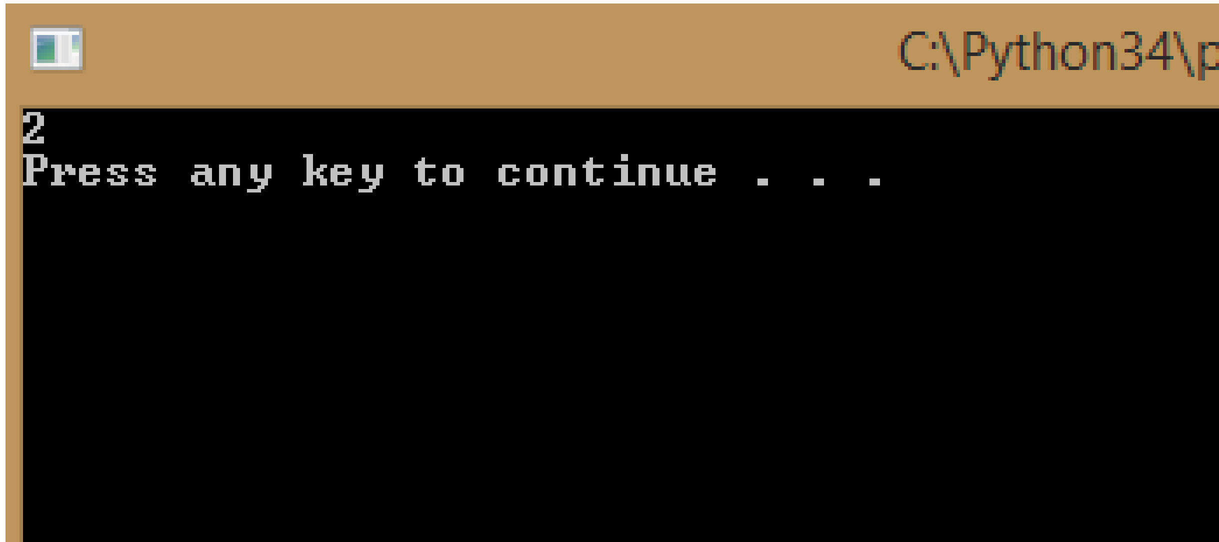


```
Susan
Press any key to continue . . .
```

What if I want to check if a particular value is in the list?

The index() function will search the list and tell you the position where the value was found

```
guests = ['Christopher','Susan','Bill','Satya']

#this will return the position in the list
#where the name Bill is found
print(guests.index('Bill'))
```



```
2
Press any key to continue . . .
```

What do you think will happen if we search for a value that doesn't exist in the list?

```python
guests = ['Christopher','Susan','Bill','Satya']

#this will return the position in the list
#where the name Bill is found
print(guests.index('Steve'))
```

The code crashes!

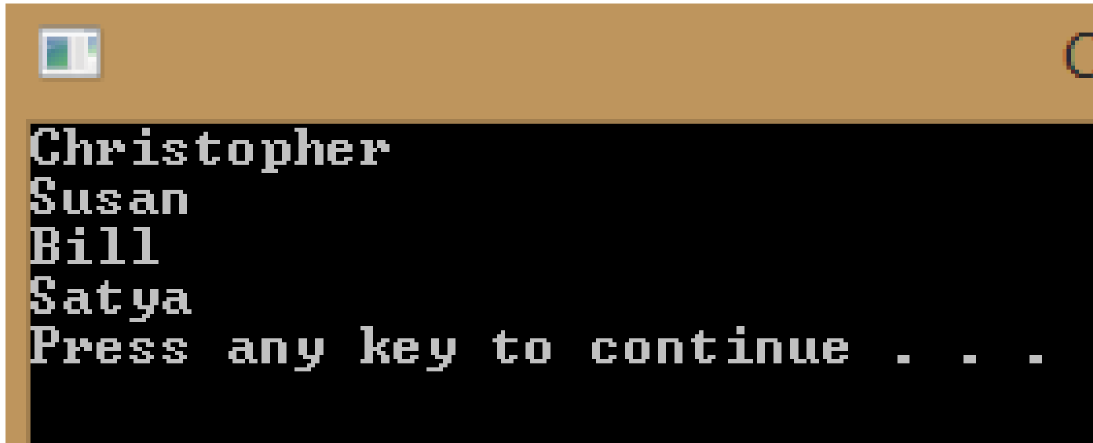
We need to add error handling

Or find another way to go through the list and find a value

What if I want to see all the values in the list?

Use a loop!

```
guests = ['Christopher','Susan','Bill','Satya']

#Create a loop that executes four times
#Since we have four values
for steps in range(4) :

     #Remember the value of steps goes up by one
     #Each time the loop executes
     print(guests[steps])
```

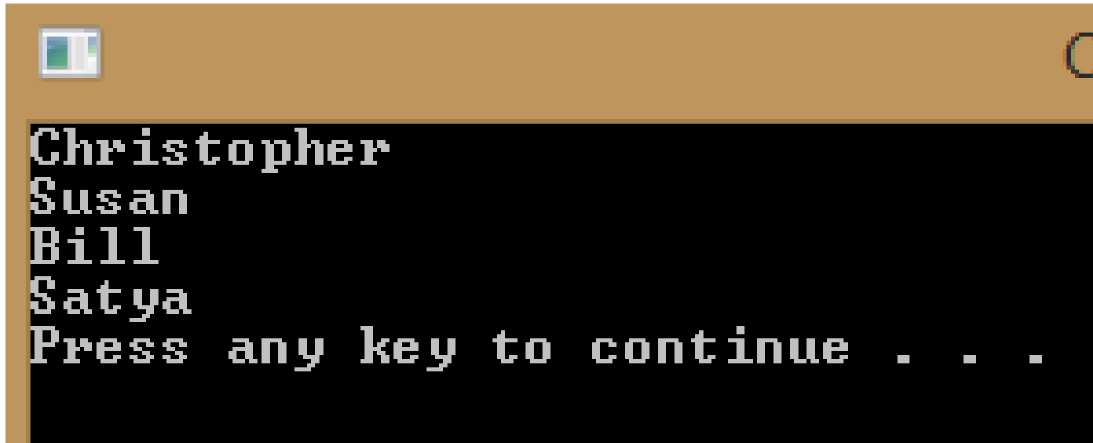


```
Christopher
Susan
Bill
Satya
Press any key to continue . . .
```

What if I don’t know how many values are in the list?

Use the len() function to find out how many entries are in your list

```
guests = ['Christopher','Susan','Bill','Satya']

#Find out how many entries are in the list
nbrEntries = len(guests)

#Create a loop that executes once for each entry
for steps in range(nbrEntries) :
     print(guests[steps])
```

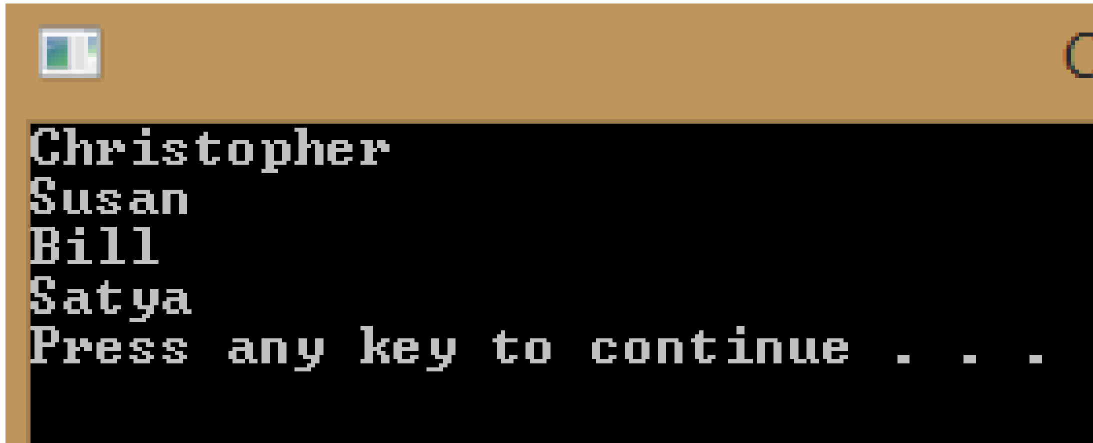


```
Christopher
Susan
Bill
Satya
Press any key to continue . . .
```

Shhhh, don’t tell anyone but there is an even easier way to go through all the items in a list

You can just tell the for loop to go through your list!

```
guests = ['Christopher','Susan','Bill','Satya']

#specify the name of your list and a variable name
#to hold each entry as you go through the loop
for guest in guests :

    #the variable guest will contain the values
    #as we go through the loop
    print(guest)
```

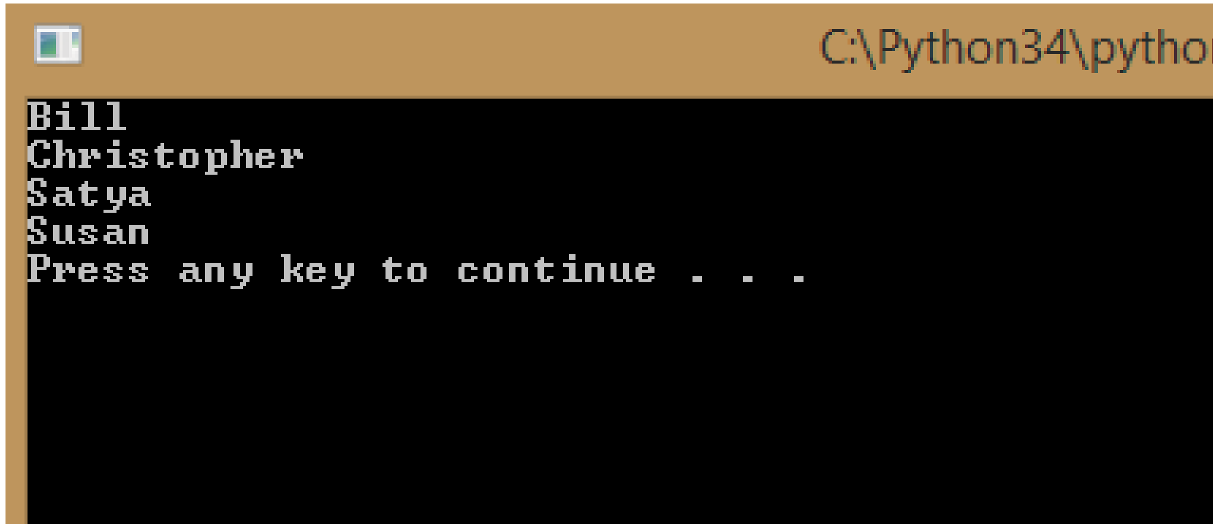


```
Christopher
Susan
Bill
Satya
Press any key to continue . . .
```

Want to sort your list?

You can sort your list with the sort() function

```python
guests = ['Christopher','Susan','Bill','Satya']

#Sort the names in alphabetical order
guests.sort()

#print the list
for guest in guests :
      print(guest)
```



```
Bill
Christopher
Satya
Susan
Press any key to continue . . .
```

MVA Microsoft Virtual Academy

Let’s put it all together!

Microsoft

Here’s a programming challenge!

- Ask the user to enter the names of everyone attending a party
- Then return a list of the party guests in alphabetical order

Ask the user to enter the names of everyone attending a party

- What command do we use to ask a user for a value?
  - input function
- What type of variable will we need to store all the names?
  - A list
- How can I ask the user for more than one name?
  - Use a loop

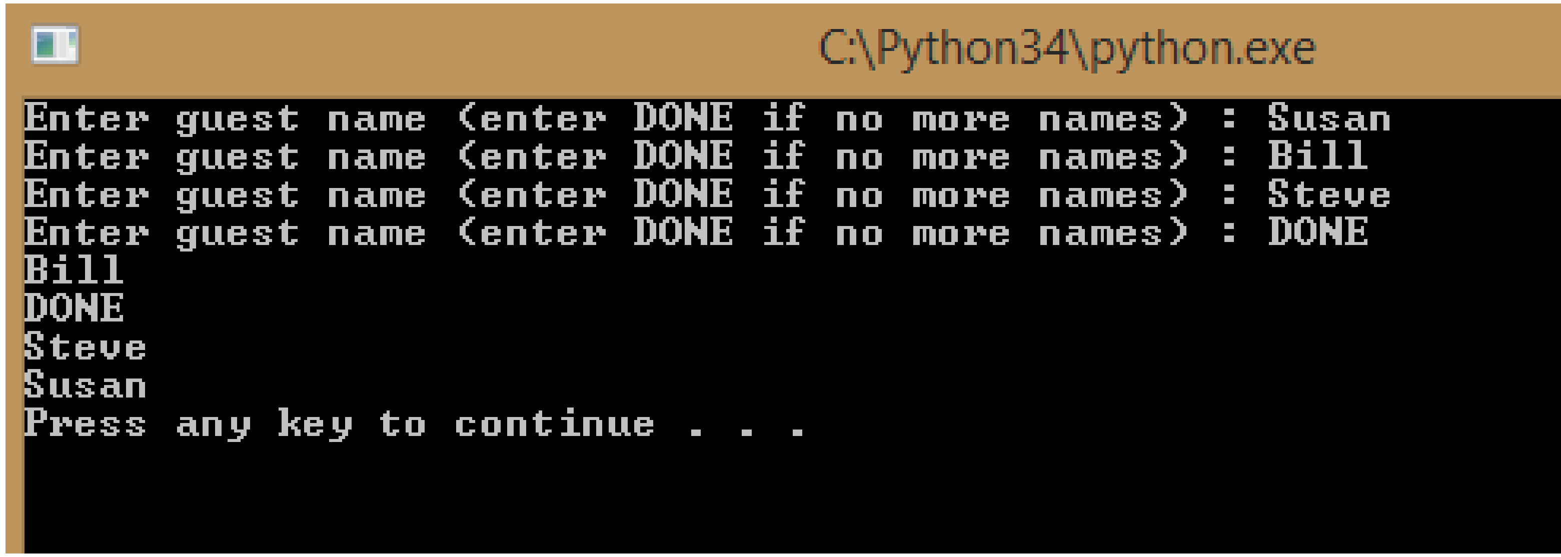
Should we use a for loop or while loop?

- Do you know how many names the user will enter?
  - No, that means we don't know how many times the loop needs to execute, so we should use a while loop
- How will the loop know when to stop executing?
  - We could have user enter a special keyword when they are done (as long as we tell them to do it!)

So… something like this?

```
guests = []
name = "  "

while name != "DONE" :
    name = input("Enter guest name (enter DONE if no more names) : ")
    guests.append(name)

guests.sort()
for guest in guests :
    print(guest)
```

We are close but our code added the name DONE to our list of guests
How can we tell the program that if the name is “DONE” not to add it?

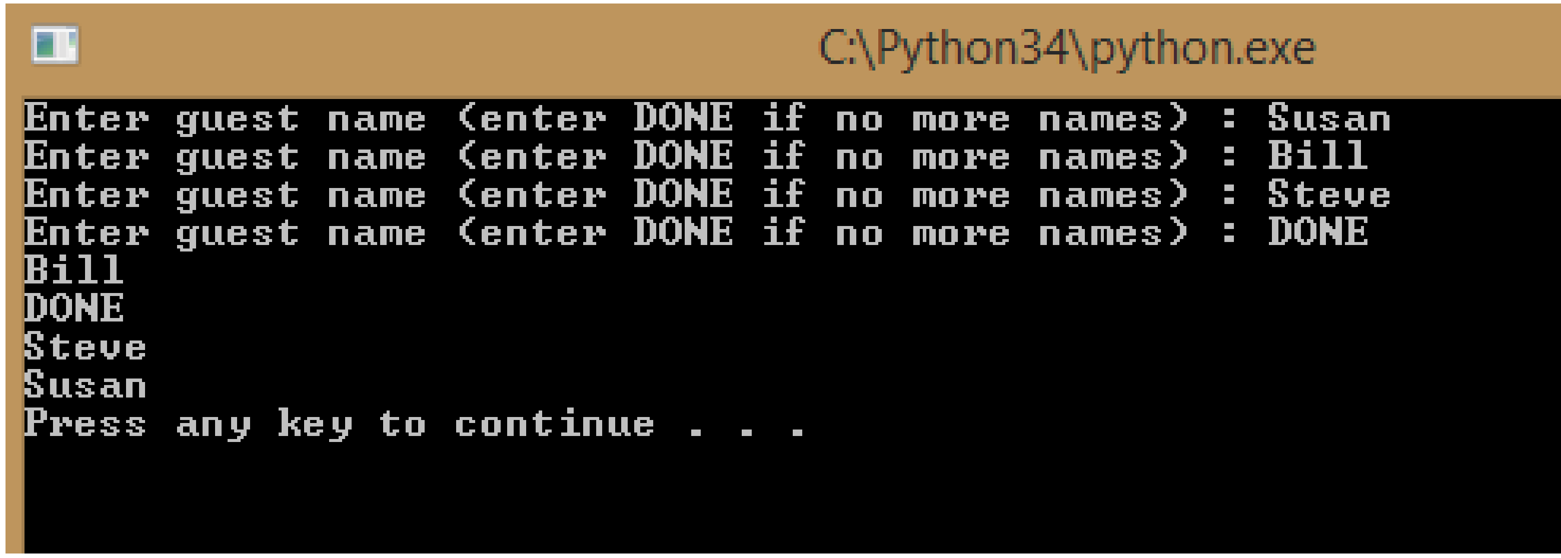


```
Enter guest name (enter DONE if no more names) : Susan
Enter guest name (enter DONE if no more names) : Bill
Enter guest name (enter DONE if no more names) : Steve
Enter guest name (enter DONE if no more names) : DONE
Bill
DONE
Steve
Susan
Press any key to continue . . .
```

Use an if statement! Awesome now we are pulling everything we learned together!

```
guests = []
name = "  "

while name != "DONE" :
     name = input("Enter guest name (enter DONE if no more names) : ")
     guests.append(name)

guests.sort()
for guest in guests :
     print(guest)
```

We are close but our code added the name DONE to our list of guests
How can we tell the program that if the name is “DONE” not to add it?


C:\Python34\python.exe

```
Enter guest name (enter DONE if no more names) : Susan
Enter guest name (enter DONE if no more names) : Bill
Enter guest name (enter DONE if no more names) : Steve
Enter guest name (enter DONE if no more names) : DONE
Bill
DONE
Steve
Susan
Press any key to continue . . .
```