Integers

Practice
active learning.
Make full of of this space by writing down observation and notes.

Objectives
- Running Python interactively
- Print integers
- Use integer operators
- Understand operator precedence
- Variables with integer values
- Write comments
- Capturing integer value from keyboard
- Using IDLE to write Python programs
- Use IDLE to open a program.

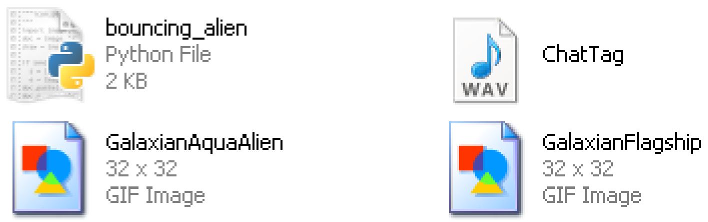
Introducing the bug

(Before going ahead with the notes, go to our web site and look for the notes. Next to the notes on "Integers" you will see "Bouncing Alien"). Click on the link and save the file to your Desktop. The file is a zipped folder. You will need to unzip it.)

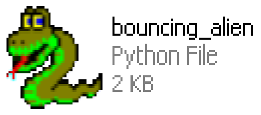
We've caught an alien bug and kept it in a box. It's desperately bouncing around trying to escape. To see it follow these instructions ...

You're given a folder called bouncing_alien:

(you will be told how to get the folder). Go into the folder by double-clicking on it. Inside the folder you will see the following four files:



ASIDE: If you installed an older version of Python, you will see this icon for a Python program:

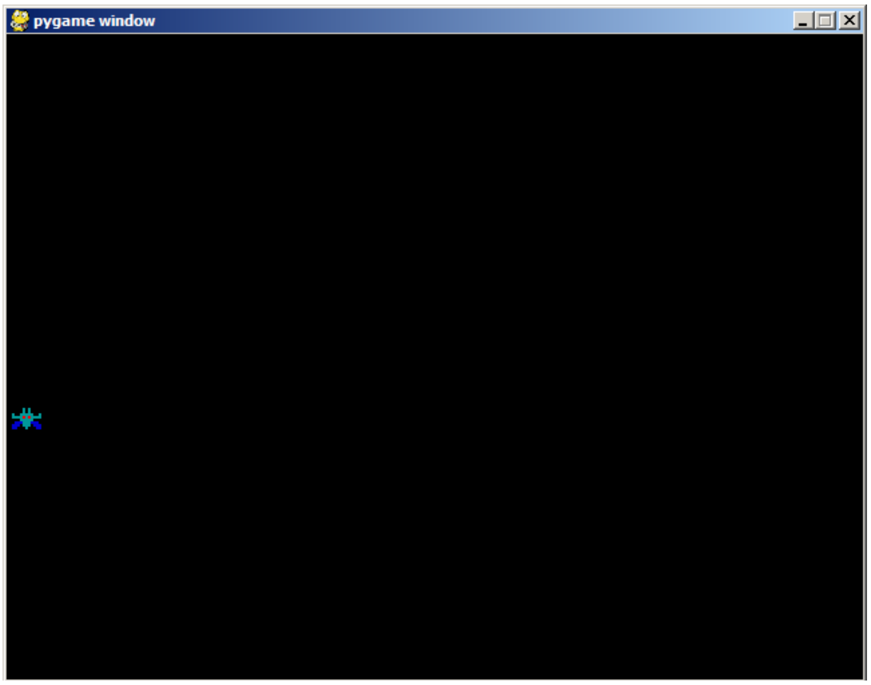


This is a Python program:

The file ChatTag.wav

is a sound file. The remaining two files GalaxianAquaAlien.gif and GalaxianFlagship.gif are images files. Try double-clicking on all of them.

A program is a bunch of instructions to the computer kept in a file. The actual name of the file of this program is bouncing_alien.py To run this program, double-click on the icon. You will see an alien bug in the box:



Phew ... an alien that size can't possible do much harm to humans ...

OK. Forget about the silly story line.

The above program bouncing_alien.py is a (relatively simple) Python program with graphics, animation and sound. By the end of this set of notes, you will learn how to view the program that produced the graphics, animation and sound. You will learn some “parameters” that control the program and how to modify them. These are called **integer variables**. Later you will learn how to draw the bug, how to make it move, how to make it bounce off a wall, how to play a sound, etc. So we have a lot of stuff to cover. Let's get started ...

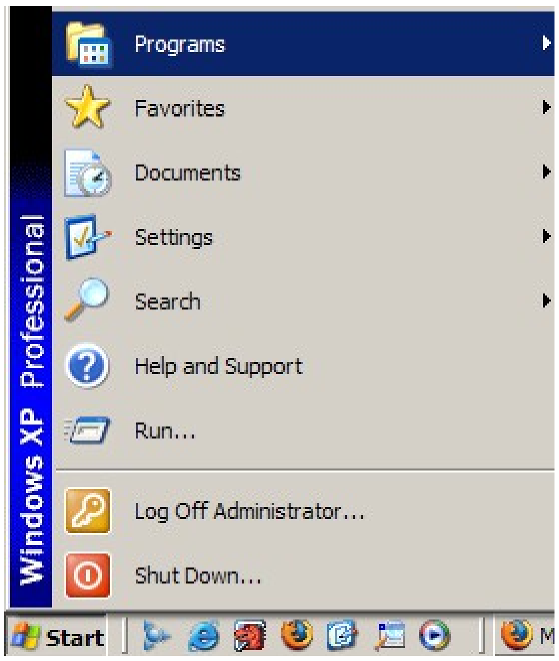
Running Python Interactively

Ready? Remember to learn actively: Don't just stare at the notes. **Follow the notes by typing, running and understanding all programs.**

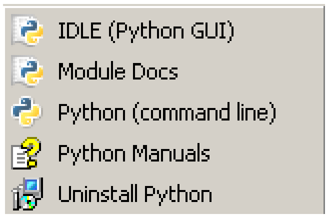
Before I begin showing your programs, I'll just give you an advice to minimize the amount of pain. **Type in whatever is shown as is.** Do not change the case (uppercase to lowercase or lowercase to upper) and do not insert or remove spaces.

OK. Here we go. First I'll show you how to run IDLE. This is a program that will allow you to execute Python commands and to write Python programs.

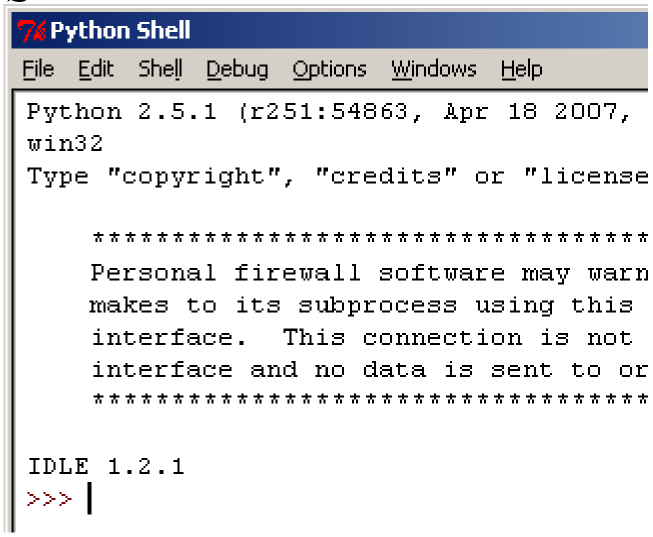
Click on the "Start" on the bottom left of your screen, select "Programs":



(Your Windows desktop might look slightly different). Look for "Python 2.5" and click on it and you should see this:



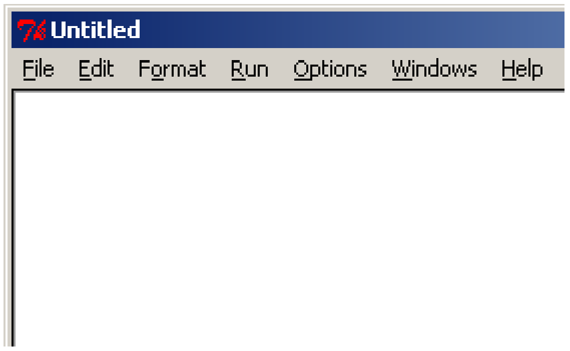
Click on “IDLE (Python GUI)” and you will get this window titled **Python Shell**:



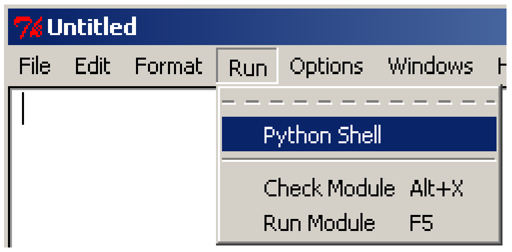
```
Python 2.5.1 (r251:54863, Apr 18 2007,
win32
Type "copyright", "credits" or "license

    ***********************************
    Personal firewall software may warn
    makes to its subprocess using this
    interface.  This connection is not
    interface and no data is sent to or
    ***********************************

IDLE 1.2.1
>>> 
```

Before I continue, in the future, instead of saying all the above to run IDLE I might say “Start > Programs > Python 2.5 > IDLE (Python GUI)”.

If you do not get the above, don't panic. You probably have this window instead:



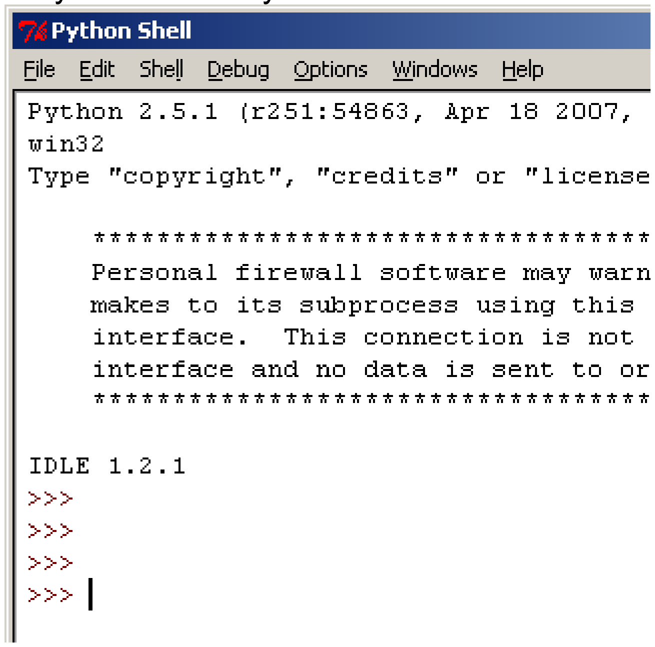
If that's the case, to get the above Python shell, just pull down “Run” and select “Python Shell”:



and you will get the above "Python Shell" window. In the future, instead of saying that I will say: "Do Run > Python Shell".

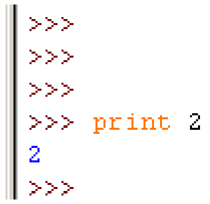
The program you just ran is called IDLE. At the top of the window you see "Python Shell". This means that IDLE allows you to "talk" to Python by give it one command at a time. A Python "command" is called a **statement**.

First press your "enter" key several times:



```
Python 2.5.1 (r251:54863, Apr 18 2007,
win32
Type "copyright", "credits" or "license

    ***********************************
    Personal firewall software may warn
    makes to its subprocess using this
    interface.  This connection is not
    interface and no data is sent to or
    ***********************************

IDLE 1.2.1
>>>
>>>
>>>
>>> 
```

Now type in `print 2` and press the "enter" key. You should get this:



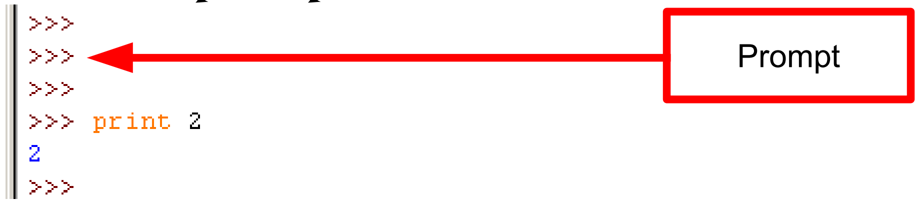
```
>>>
>>>
>>>
>>> print 2
2
>>>
```

Type "print 2" and then press the "enter" key.

Woo hoo ... !!! You've just commanded the Python shell to print the number 2.

Nowadays, most people interact with programs by clicking on icons with the mouse. The Python shell interacts with us through a **command line interface**. You don't see pictures or buttons. You enter text and the program responds with text (or graphics or sound ...)

The “>>>” is called a **prompt**:



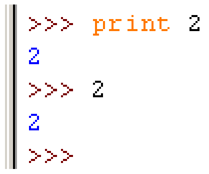
```
>>>
>>>
>>>
>>> print 2
2
>>>
```

When you see the prompt, it means the Python shell is ready to receive a statement to execute.

Note also that when you pressed enter three times, the shell doesn't do anything. So it's OK to issue an “empty” statement.

Instead of saying “Python shell prints 2” or “IDLE prints 2”, from now on I might just say “Python prints 2”.

Actually to get Python to print 2 you can also type “2” and then press the “enter” key:

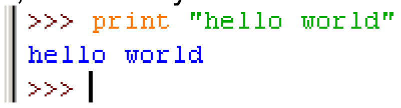


```
>>> print 2
2
>>> 2
2
>>>
```

Exercise. You have 10 seconds to close IDLE, open it again, and write a Python statement that prints the number 42.

Hello World and Strings

Before we go on, I must let you run the following program:



```
>>> print "hello world"
hello world
>>> 
```

It's a tradition among programmers to run a hello world program the first time they learn a new language. The practice has been around since 1974.

Let's compare the statements:

```
    print 2
```

and

```
    print "hello world"
```

The print statement allows you to print values to the shell's window. While **2** is a whole **number**, **"hello world"** is a **string**. Think of a string as textual data for the time being.

The important thing to remember is that you **can't throw aways the quotes in a string**.

One way to learn a new language is through drills. This is especially so when you learn a new language.

Now try to get the Python shell to execute this statement:

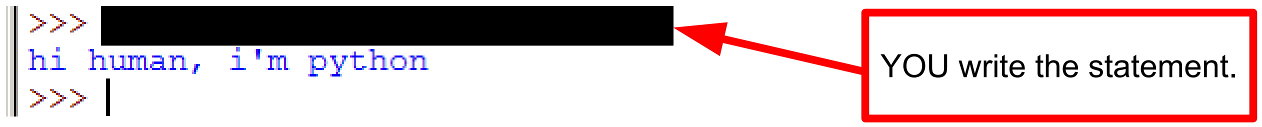
```
>>> print "hi human"
```

Exercise. Don't believe everything you hear! Type in the following. Note that there are no quotes around hello world.

```
>>> print hello world
```

What happens when you press enter?
Answer: _____

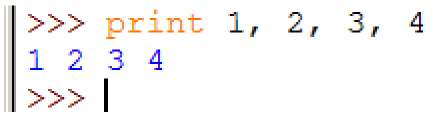
Exercise. Now it's your turn. Write a Python statement that will make Python say this



```
>>> [redacted]
hi human, i'm python
>>> 
```

Printing more than one thing

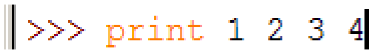
Try this



```
>>> print 1, 2, 3, 4
1 2 3 4
>>> 
```

This tells you that you can print more than one value.

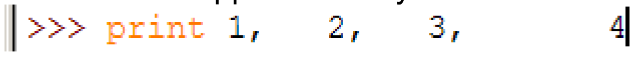
Exercise. Can we leave out the commas in the above statement? In other words what happens when you run this:



```
>>> print 1 2 3 4
```

Answer: _____

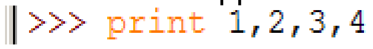
Exercise. Can you insert more spaces after the commas? In other words what happens when you run this:



```
>>> print 1,    2,    3,         4
```

Answer: _____

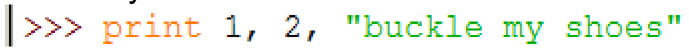
Exercise. Can you remove the spaces after the comma? In other words what happens when you run this:



```
>>> print 1,2,3,4
```

Answer: _____

Exercise. Can you you mix strings and integers in a print statement? Try this:



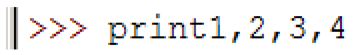
```
>>> print 1, 2, "buckle my shoes"
```

Answer: _____

Exercise. What if you execute the above statement but with extra spaces between the words `buckle` and `my`?

Answer: _____

Exercise. Can you leave out the space after `print`? Try this:



```
>>> print1,2,3,4
```

Answer: _____

Exercise. What if you have spaces before the word `print` like this:

```
>>>    print 1, 2, 3, 4
```

Does it work?

Answer: _____

Exercise. What about this? (i.e. `Print` instead of `print`):

```
>>> Print 1,2,3,4
```

Answer: _____

LISTEN VERY CAREFULLY! The above “experiments” based on minor modifications of the print statement

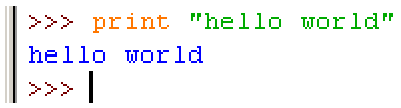
```
print 1, 2, 3, 4
```

is what we call **active learning**. By experimenting with what you know, you are actively involved with the new knowledge and so it's easier to remember what you learned. Even more importantly, active learning allows you to go beyond what you know. In fact the skill of **learning to learn** through active learning is just as important as the knowledge you receive in your classroom or from books, important as it may be. So remember to experiment and to question what you learn from books, from classrooms, etc.

OK. Let's go on ...

Integers and Some Operators

From now on instead of screen shots of Python shell such as:



```
>>> print "hello world"
hello world
>>> 
```

I will show a text box like this:

```
>>> print "hello world"
hello world
>>>
```

OK, let's go on ...

You can use the Python shell as a calculator. Try to add 12 and 4:

```
>>> print 12 + 4
16
>>> 12 + 4
16
>>>
```

Now try this. Note that there are no spaces before and after +:

```
>>> print 12+4
16
>>> 12+4
16
>>>
```

Exercise. What is 135 + 246? First do this by hand. Next verify using Python.

Answer: 135 + 246 = _____ (do this by hand)

Answer: 135 + 246 = _____ (do use Python)

Exercise. Try to get Python to subtract 12 from 36. Verify by hand.

Answer: 36 – 12 = _____ (do this by hand)

Answer: 36 – 12 = _____ (now use Python)

Exercise. Try to get Python to multiply 12 and -12. Verify by hand.

Multiplication is * just like your TI calculators

Answer: 12 * – 12 = _____ (by hand)

Answer: 12 * – 12 = _____ (from Python)

Exercise. Now compute 1 + 2 * 3 – 4 by hand. Verify using the Python. Does your answer agree with Python?

Answer: 1 + 2 * 3 – 4 = _____ (by hand)

Answer: 1 + 2 * 3 – 4 = _____ (from Python)

2 is a value while 2 + 3 * 4 is an **expression**. Computing the value of 2 + 3 * 4 is called **evaluating** the expression

Exercise. Python understands parentheses. Get Python to tell you what is (1 + 2) * 3 – 4.

Answer: (1 + 2) * 3 – 4 = _____ (from Python). Check with someone.

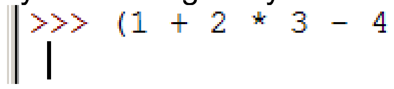
IMPORTANT JARGON. When you see something like 1 + 2 * 3, you need to decide whether to perform the + or the * first. Math people have decided long time ago that * and / comes before + and -. Such a rule is called an **operator precedence rule**. It's important to know that it's not by magic that Python evaluates expressions just like the way we do math. It's because the people who wrote Python intended it that way.

+, -, * are called **operators**

How to say “Oops” to Python

What happens when you try to get Python to evaluate (1 + 3 * 3 – 4? Note that the right parenthesis is missing!

The Python shell gives you this:

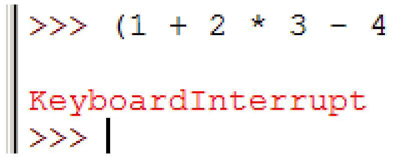


```
>>> (1 + 2 * 3 - 4
 
```

Python knows that the statement is incomplete and is waiting for you to complete it. You can press enter as many times as you like. Python stubbornly waits for you to complete your expression.

Here's what you can do to stop the Python shell from waiting for you to finish up what you want to say – when in fact you've made a mistake and wanted to stop. Do a Ctrl-C. You should see this:

“Ctrl-C” means to hold the “ctrl” key down and press “c” and then release both.

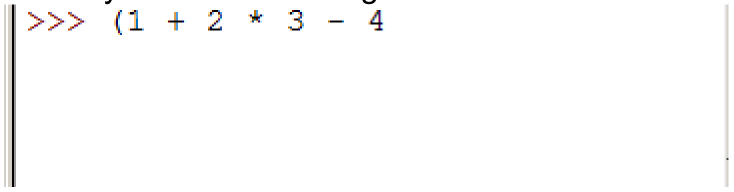


```
>>> (1 + 2 * 3 - 4

KeyboardInterrupt
>>> 
```

And you get the prompt back and the incorrect statement is not executed.

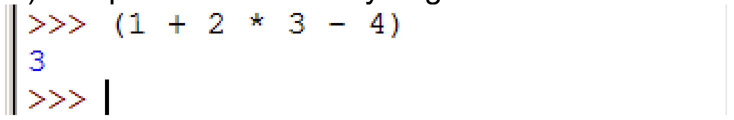
Another way would be to press backspace to wipe out the incorrect statement or make corrections.

Exercise. At the shell prompt, type in (1 + 2 * 3 – 4 and press the enter key several times to get:



```
>>> (1 + 2 * 3 - 4
```

Using the backspace key, correct the statement to get (1 + 2 * 3 – 4) and press the enter key to get:



```
>>> (1 + 2 * 3 - 4)
3
>>> 
```

Finally if you make a mistake, you can close IDLE and run it all over again. This is of course very slow (not to mention very embarrassing).

Integers and More Operators

There are three other operators I want to talk about.

Try to get Python to evaluate 3/2 and you get

```
>>> 3/2
1
>>>
```

/ is the quotient operator

Yikes!!! Is Python stupid or something? Shouldn't 3/2 be 1.5???

Now try 9/4. Try a few more examples. Can you guess what's really happening?

Exercise. What is 1/0?

Answer: _____ (your answer)

Answer: _____ (Python's answer)

Exercise. Now try 3%2. What do you get? Try 9%4. What do you get? Try 8%4. What do you get?

3%2 Answer: _____

9%4 Answer: _____

8%4 Answer: _____

The **%** gives you the **remainder**. 9%4 is 1 because the remainder after dividing 9 by 4 is 1.

** is the exponentiation operator

Exercise. Complete this table:

2**1 Answer: _____

2**2 Answer: _____

2**3 Answer: _____

2**4 Answer: _____

3**2 Answer: _____

So tell me what is ** for?

Exercise. What is 1234**5678? (There's an "L" at the end of the answer. Just ignore it.) Now let's see you do that with your TI calculator!!!

Answer: 1234**5678 = _____ (don't bother writing!)

Exercise. Evaluate 1 + 2 * 3 ** 4 – 5 % 6 by hand. Check with Python.

IMPORTANT JARGON: Whole numbers (both positive and negative) are called **integers**.

Exercise. The time is now 3:35PM. After working on his Python notes for 134 minutes, how many minutes does John have to wait till dinnertime at 6PM?

Summary (DIY)

For integer values we have the following:

	Explanation	Example
a + b	Sum of a and b	1 + 2 is 3
a – b	Difference of a and b	1 – 2 is -1
a * b	Product of a and b	2 * 3 is 6
a / b	Quotient of a by b	5 / 2 is 2
a % b	Remainder of a by b	5 % 3 is 2
a ** b	a to the power of b	5**2 is 25
-a	Negative of a	

Integer operators in Python follow PEMDAS. Remember PEMDAS?

P	Parentheses
E	Exponentiation (i.e. power)
M	Multiplication
D	Division
A	Addition
S	Subtraction

Here's the precedence rules for Python:

()	Parentheses
**	Exponentiation
*, /, %	Multiplication, quotient, remainder
+, -	Addition, subtraction

Operators at the same level are evaluated left to right. For example, if you look at this:

```
1 – 2 * 3 / 2 + 1
```

* and / goes first. But they are at the same level. So 2 * 3 / 2, we work from left to right. So we do * first. This gives 6 / 2 which is 3. So we get

```
1 – 3 + 1
```

Now – and + are at the same level. So we go from left to right again. This gives -2 + 1 which is -1.

Integer Variables

If Python is only used as a calculator, then we should probably forget about it! Thank goodness it's a lot more than that.

Try this:

```
>>> x = 1
>>> print x
1
>>>
```

We've just created a **variable** `x`.

A variable in Python is very similar to the variable you know from Mathematics.

However ... **LISTEN UP!!!** ... the "=" sign above is different from the "=" sign in Mathematics.

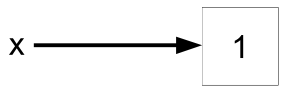
In your Math classes, when your teacher tells you to solve the equation

$$4x + 1 = 2x – 5$$

he's/she's telling you to find a value for x so that the left and right hand side of the equation are the same. (By the way, can you solve the above equation???)

In Python (and also for many other programming languages), "=" means "**give the value on the right to the variable on the left**". So `x = 1` in Python means, create a variable `x` and give it a value of 1.

At this point, it's helpful to have a picture to help you understand variables. When you execute `x = 1`, think of this picture:

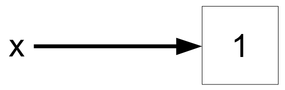


You can think of Python as keeping this picture in its brain. So a variable is like a name of a box where you can put values.

Try this:

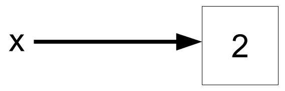
```
>>> x = 1
>>> print x
1
>>> x = 2
>>> print x
2
>>>
```

This says that you can **change** the **value** of `x`. This is how you think of Python executing the statements. After `x = 1`, Python remembers this



Of course the second statement `print x` prints the value of `x`. To do that Python now looks at the value of `x`, and says "OK, I need to print 1" and prints it to the window.

Next Python execute the third statement, `x = 2`, and gives `2` to `x`. At this point Python remembers this in its brain:



Got it? The important thing to note is that the value of `x` now is `2`, not `1`. Python does not remember the old value of `x`.

Exercise. Give `x` the value `1 + 2`. Print `x`. What is the value of `x`? Now verify using Python.

Answer: ______ (your answer)

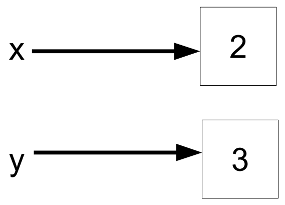
Answer: ______ (Python's answer)

The above exercise tells you that besides giving a value to a variable, you can give an expression (example: `1 + 2`) to a variable. Python will figure out the value (example: `1 + 2` is `3`) and give it to the variable.

Try this:

```
>>> x = 2
>>> print x
2
>>> y = 3
>>> print y
3
>>> print x
2
```

This tells you that Python **remembers** values of **all** variables. In other words Python can remember more than one variable. This is what Python remembers at end of these statements:



Easy right?

Exercise. Continuing from the above type in the following:

```
>>> x = 2
>>> print x
2
>>> y = 3
>>> z = x * y
>>>
```

First guess the value of `z`.

Answer: `z` has value _____ (your guess)

Next do a “`print z`” to get Python to tell you the value.

Answer: `z` has value _____ (from Python)

This example tells you that you **can use operators on variables** (duh – it's only reasonable right?). Remember that "=" means give the value on the right to the variable on the left. So in this case, you have to evaluate "`x * y`" to get a value before you can give it to `z`. Now, it's IMPORTANT to note that while computing "`x * y`", the value of `x` is 2 and not 1.

Another Picture

Before we go on I want to give you a picture that will help you visual how Python computes.

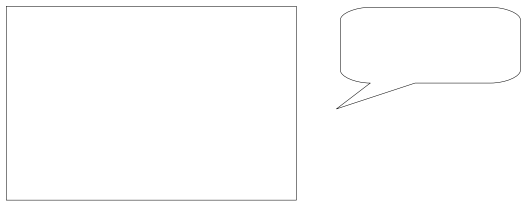
Something like this is called a model

I'll use the above example. There are 4 statements:

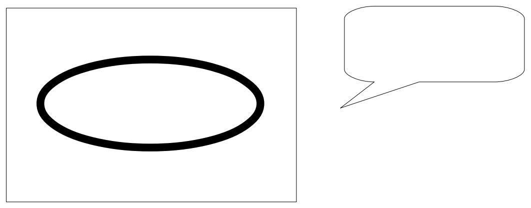
```
x = 2
print x
y = 3
z = x * y
```

Python will execute them one at a time from top to bottom (well ... at least for now!)

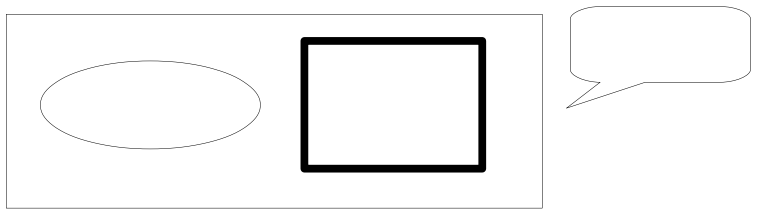
Think of this box as Python. Whenever you give a print statement, something appear in the speech bubble.



You now know that Python can do math. Imagine a part of Python is involved in performing the +, -, *, /, %, ** operations. So here's a picture of Python with its brain for doing math:



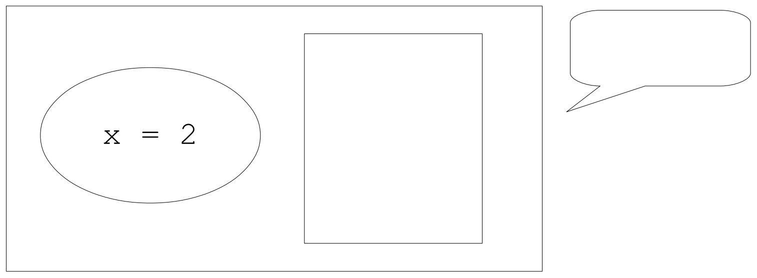
You also know now that Python remembers variables and their values. So think of Python as having a memory:



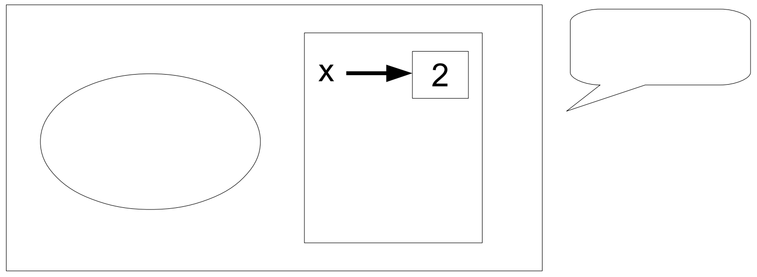
So let's begin simulating Python with statements:

```
x = 2
print x
y = 3
z = x * y
```

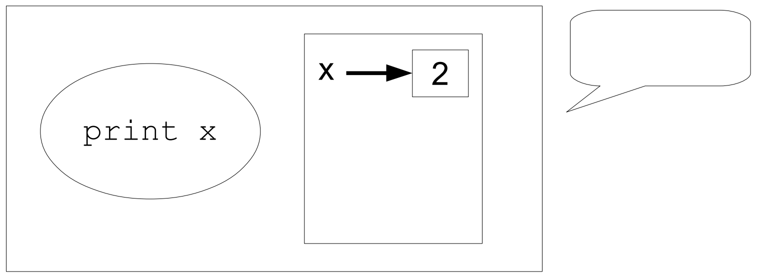
First Python read the first line of the program:



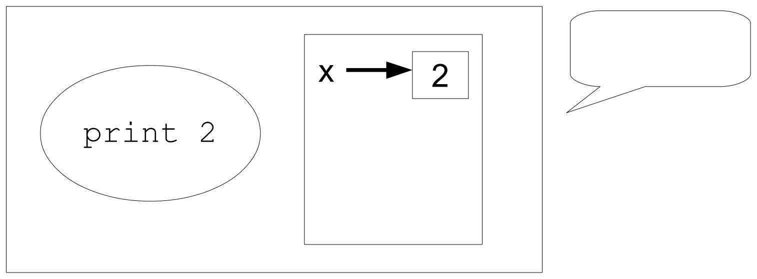
After executing `x = 2`:



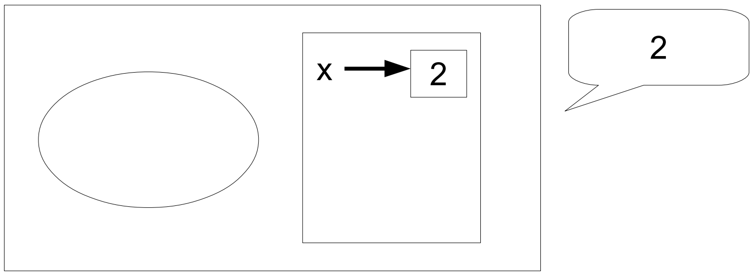
Python reads the next statement:



and realizes that he (it?) needs the value of x. Finding the value of x,



Python prints 2:

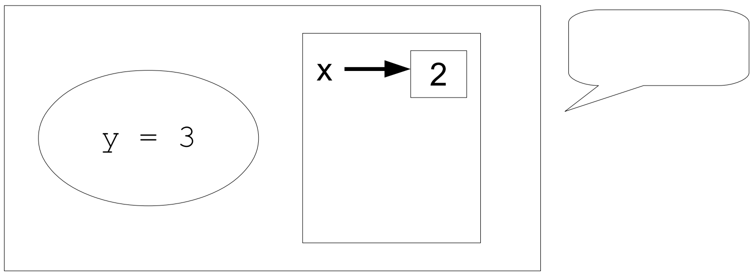


Note that the **program** is not changed. It is still

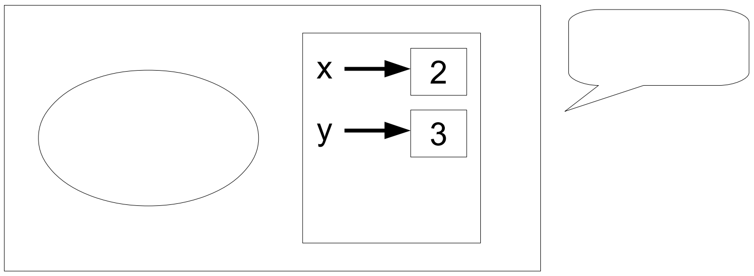
```
x = 2
print x
y = 3
z = x * y
```

In other words the second line is `print x` and not `print 2`.

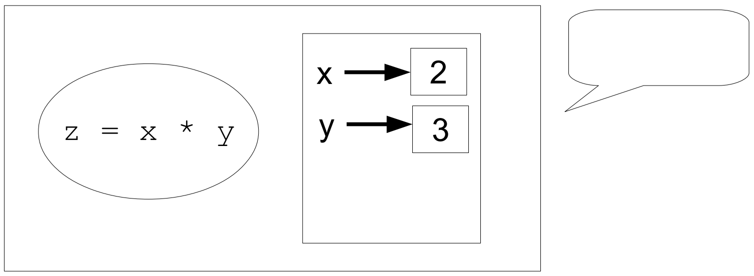
Python then reads the third statement:



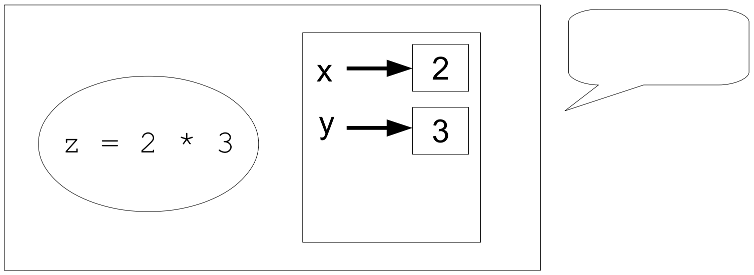
and executes it:



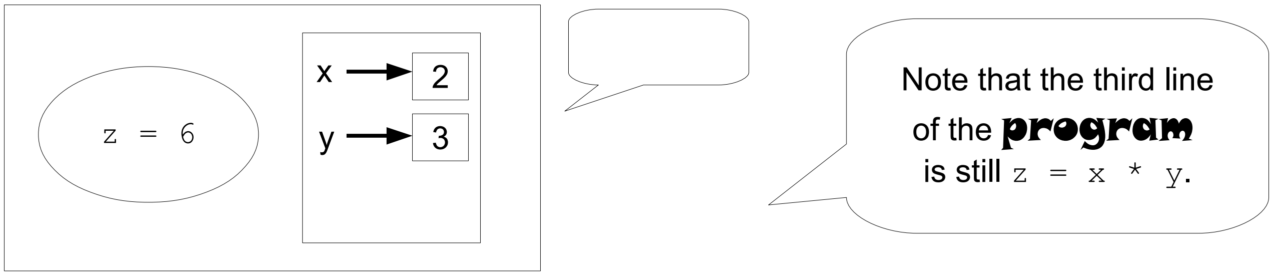
Now we want to execute `z = x * y`:



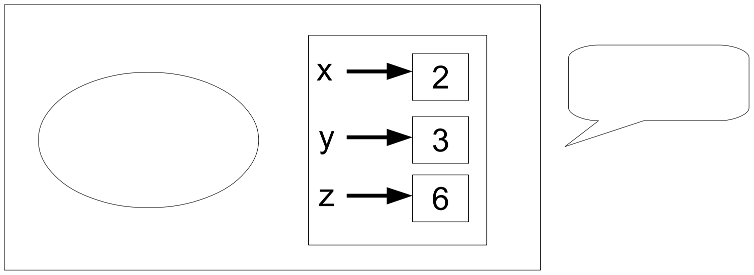
The thing to remember is that first Python has to compute the value on the right. This is `x * y`. Forget about = for the time being. First Python goes into its memory to look for `x` and say "AHA! `x` is 2". Next it looks for the value of `y` in its memory; it's 3. This is what Python is thinking about at this point:



Now Python evaluates `2 * 3` and get `6`:


z = 6
x
2
y
3
Note that the third line of the **program** is still `z = x * y`.

Now it's ready to create variable `z` and give it 6:



Exercise. Can you tell me why Python will choke when you try to run the statement "`x + 1 = 3`"? (Try it)

Answer: ____

Exercise. Can you print the value of a variable that hasn't been given a value? Well ... let's just do an experiment. What happens when you type in the following and then press the enter key?

```
>>> print i
```

Answer: _____

It's important to remember that the Python shell will "start from scratch” each time you start it. In other words when you close the shell program, all the variables are lost.

Exercise. Close IDLE completely. Run IDLE and create an integer variable `x`, setting it's value to 3. Close IDLE and run it again. Try to print `x`. Does it work?

More on =

The following example is **very important**.

```
>>> x = 2
>>> x = x + 1
>>> print x
3
>>>
```

Now if you look at the statement:

```
    x = x + 1
```

You might think ... huh? In Math this is garbage because this means (after crossing out x on both sides you get):

0 = 1

But remember in Python,

```
    x = x + 1
```

means “evaluate the right to get a value and give that value to the variable on the left”. So this is how Python thinks:

- I need to evaluate `x + 1`.
- Right now `x` is 2.
- So `x + 1` is `2 + 1` which is `3`.
- So `x = x + 1` is the same as giving `3` (i.e. the value on the right) to `x` (i.e. the variable on the left).
- I need to print `x`. Now `x` is 3. So I will print 3.

Note that `x = x + 1` is the same as increase x by 1.

Exercise. Suppose `x` is already created and has an integer value. Write a statement to increase the value of `x` by `2`. Verify with Python by printing the value of x.

Exercise. What is the value of `a` after the following three statements?

```
>>> a = 3
>>> b = 5
>>> a = a + b
>>>
```

Answer: a has value _____ (your educated guess)

Answer: a has value _____ (from Python)

Exercise. What is the value of a after the following statements?

```
>>> a = 5
>>> a = a - a * a
>>>
```

Answer: a has value _____ (your educated guess)

Answer: a has value _____ (from Python)

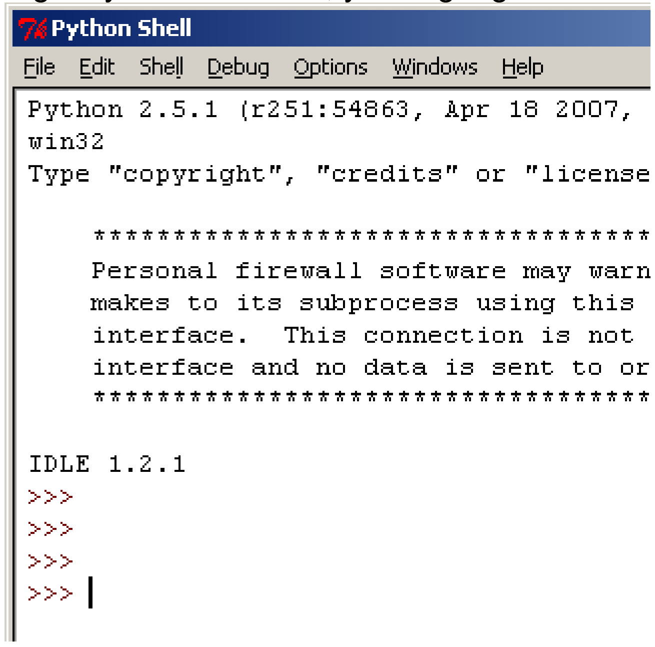
We have been printing the value of a variable. Of course it's not too surprising that you can **print an expression**:

```
>>> x = 1
>>> y = 2
>>> print x + y
3
>>>
```

Using IDLE to Write Python Programs

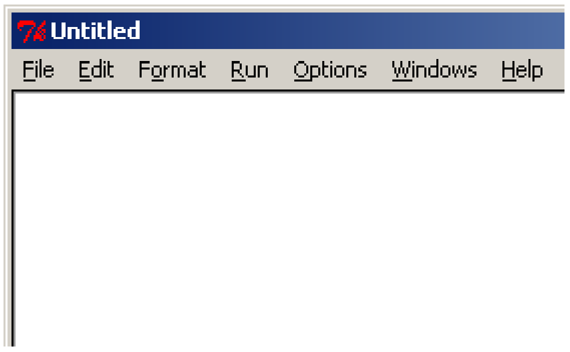
So far we have been running Python **one statement at a time**. In most cases, you want to run a program which is made up of many statements. So let's learn how to do that.

Close your Python windows and start IDLE again. Depending on your installation, you might get either this:

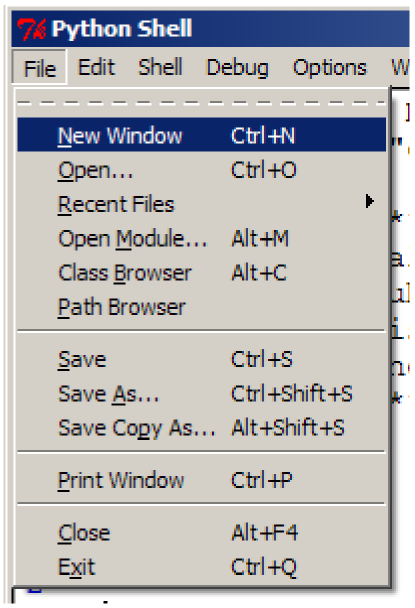


```
Python 2.5.1 (r251:54863, Apr 18 2007,
win32
Type "copyright", "credits" or "license

    **********************************
    Personal firewall software may warn
    makes to its subprocess using this
    interface.  This connection is not
    interface and no data is sent to or
    **********************************

IDLE 1.2.1
>>>
>>>
>>>
>>>
```

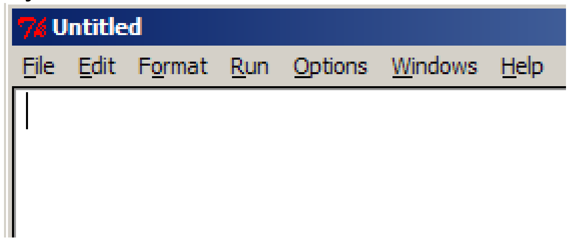
or this:



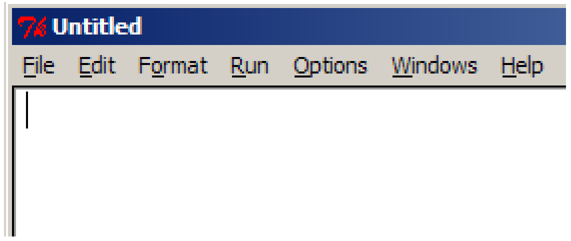
The second window (with the title “Untitled”) is where you write a program. If you have the second window you're good to go. Otherwise, if you only have the first window, you have to do this: Pull down the “File” menu



and select “New Window”. By the way, you notice that next to the “New Window” option is “Ctrl-N”. So instead of File > New Window, you could've done Ctrl-N, which is much faster. Now you have a new window:

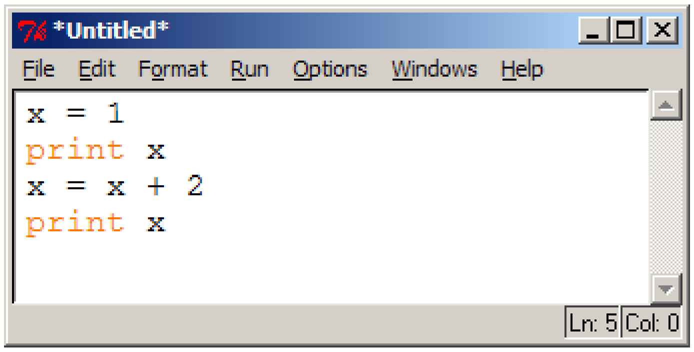


In either case you should try to get this window before continuing:



OK. Let's move on ...

The Untitled window is where you type in a **program**, i.e. **a sequence of statements.** So go ahead and type in the following program:

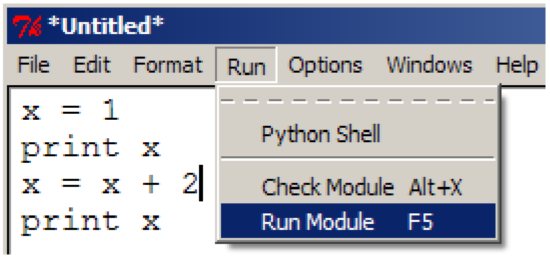


```
x = 1
print x
x = x + 2
print x
```

(I shrunk my window).

Note that the statements are **not** executed ... yet.

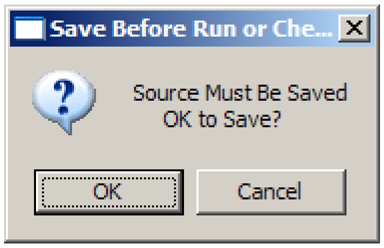
Now you want to run this program. This means you want Python to run **all** the statements one at a time until it reaches the end. On the new window, do Run > Run Module:



```
x = 1
print x
x = x + 2
print x
```

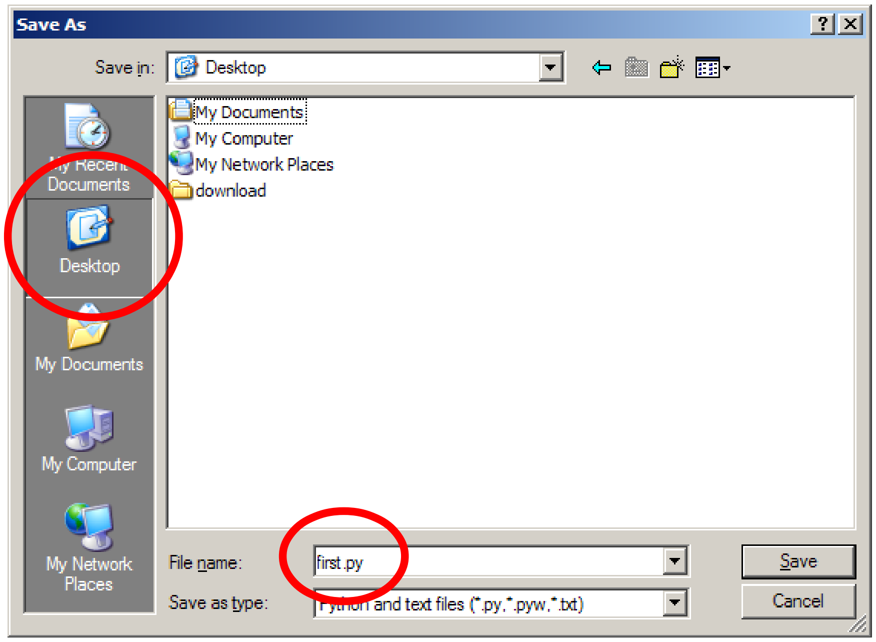
TIP: Use F5 to run your program.

IDLE will ask if it's OK to save your program:

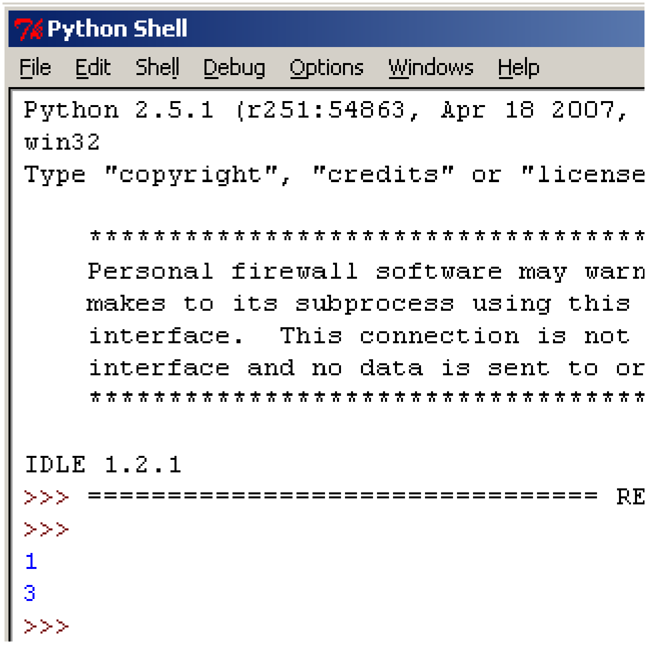


Source Must Be Saved
OK to Save?

Click on “OK” (or just press the enter key) and IDLE will ask you for a name and location for the program. Let's keep it on the “Desktop” and let's call the program “first.py”:



Click on “Save” and IDLE will save your program and run it. Note that the printout of your program is sent to the Python shell:

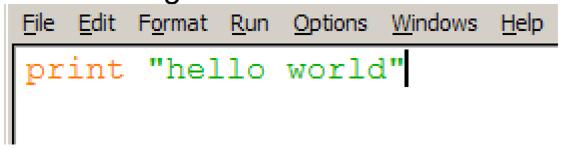


```
Python 2.5.1 (r251:54863, Apr 18 2007,
win32
Type "copyright", "credits" or "license

    ***********************************
    Personal firewall software may warn
    makes to its subprocess using this
    interface.  This connection is not
    interface and no data is sent to or
    ***********************************

IDLE 1.2.1
>>> ================================ RE
>>>
1
3
>>>
```

Go to your Desktop and make sure you see your saved Python program:

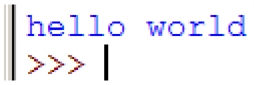
Note also that your "Untitled" window now has a different title. Check it out.

Exercise. Close IDLE. Run IDLE and create a program containing the following statement:



```
print "hello world"
```

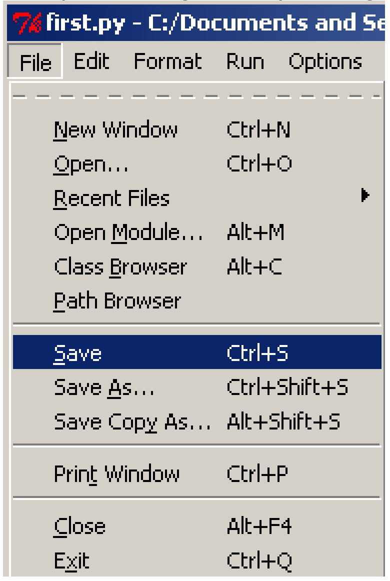
Save the program as hw.py on your Desktop. Run the program and verify that you get the following output on your shell:



```
hello world
>>> 
```

Verify that your hw.py is saved on your Desktop.

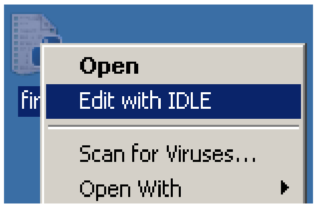
By the way, recall that when you want to run a program, IDLE will save it first. But suppose you want to manually save your program. (For instance, you're only half way through with typing a program on your laptop which is running low on battery, and you don't want to lose your work.) You can save your program by doing File > Save:



Modifying a Program

Now close your IDLE. Suppose you want to view or modify an *existing* program. How would you do it through IDLE? Let's try it out on first.py.

Go to your Desktop. Right click on first.py's icon and select “Edit with IDLE”:

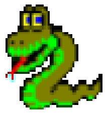


You get an IDLE window containing the program in first.py. You can then go on modifying that program.

That's it.

Whenever you have a Python program, you can see the code by doing the above. It's easy to recognize a Python program. They all have this icon:

For older Python versions, the icon is

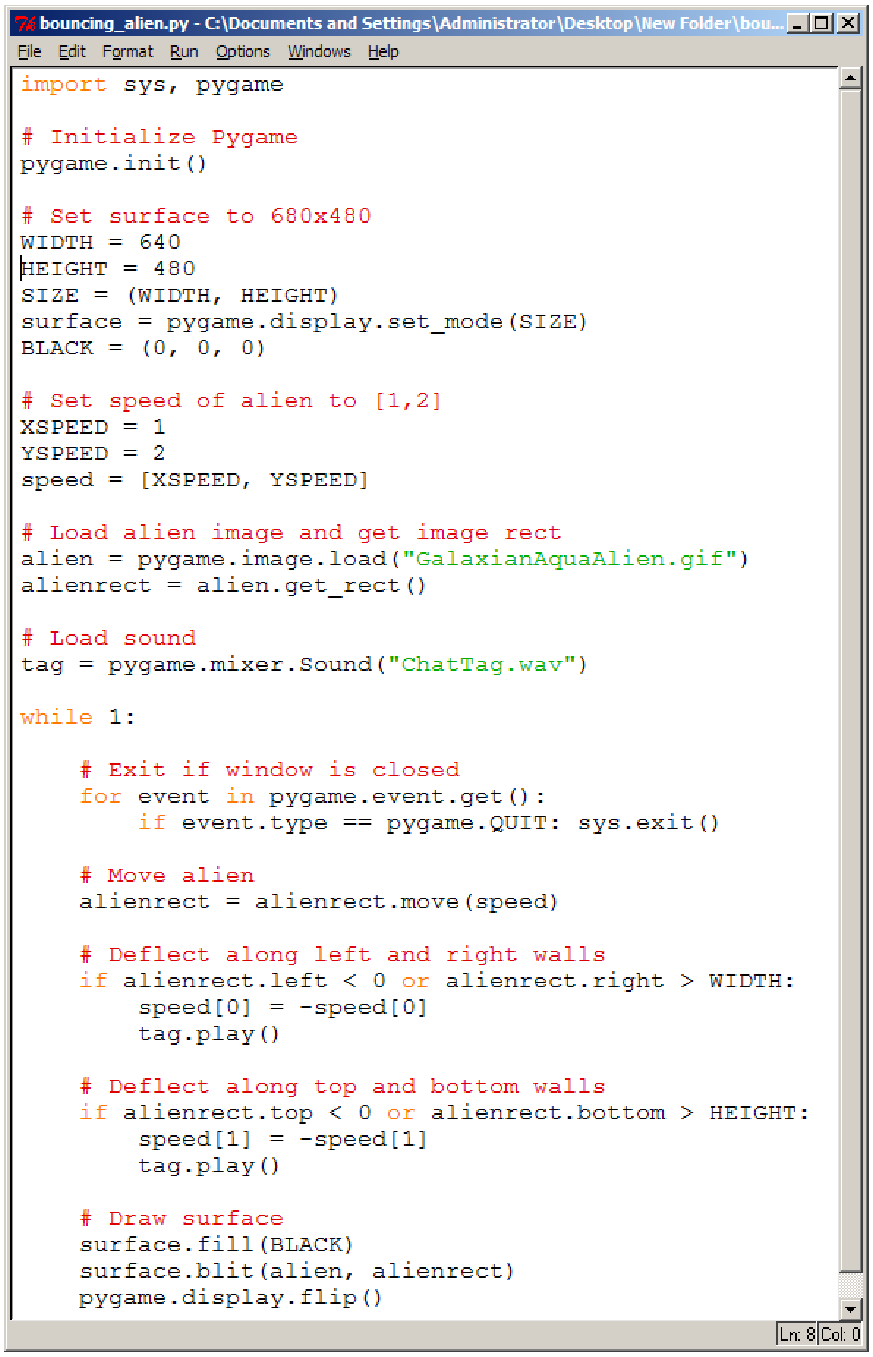


Exercise. Open hw.py from your Desktop. Change it so that when you run it, you get the following in your shell:

```
hello columbia
>>>
```

Bouncing Alien – the code

Exercise. Armed with your knowledge, open the program for the bouncing alien bug, i.e. bouncing_alien.py. You should see this:


bouncing_alien.py - C:\Documents and Settings\Administrator\Desktop\New Folder\bou...
File Edit Format Run Options Windows Help
Ln: 8 Col: 0

```
import sys, pygame

# Initialize Pygame
pygame.init()

# Set surface to 680x480
WIDTH = 640
HEIGHT = 480
SIZE = (WIDTH, HEIGHT)
surface = pygame.display.set_mode(SIZE)
BLACK = (0, 0, 0)

# Set speed of alien to [1,2]
XSPEED = 1
YSPEED = 2
speed = [XSPEED, YSPEED]

# Load alien image and get image rect
alien = pygame.image.load("GalaxianAquaAlien.gif")
alienrect = alien.get_rect()

# Load sound
tag = pygame.mixer.Sound("ChatTag.wav")

while 1:

    # Exit if window is closed
    for event in pygame.event.get():
        if event.type == pygame.QUIT: sys.exit()

    # Move alien
    alienrect = alienrect.move(speed)

    # Deflect along left and right walls
    if alienrect.left < 0 or alienrect.right > WIDTH:
        speed[0] = -speed[0]
        tag.play()

    # Deflect along top and bottom walls
    if alienrect.top < 0 or alienrect.bottom > HEIGHT:
        speed[1] = -speed[1]
        tag.play()

    # Draw surface
    surface.fill(BLACK)
    surface.blit(alien, alienrect)
    pygame.display.flip()
```

Yikes!!! Looks like Greek!!! Don't worry. We'll get there. Now close the program and let's get back to learning Python.

Difference between Shell and Program

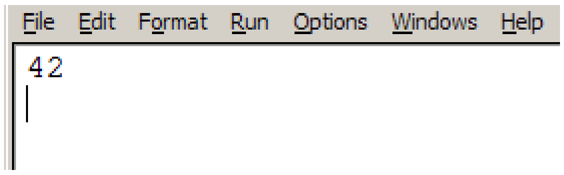
The main difference between running a statement at a time from the shell and running a program is that a program can be more than one statement. And when you **run a program**, Python will execute **all the statements** in the program one at a time from **top to bottom** (at least for now).

But there's another difference. You know that in the shell, when you enter a value or an expression such as

```
>>> 1 + 2
3
>>>
```

the shell prints the value for you. You don't have to say "`print 1 + 2`". However for a program, values are not printed. You ***have*** to use `print` to print values.

Exercise. Create a program (you choose the name of the program this time – don't forget to end with .py) with this statement:



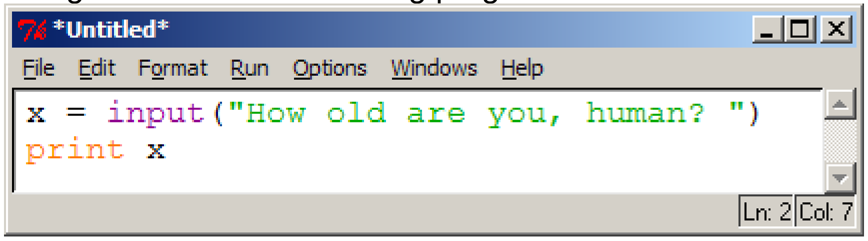
```
42
```

Run it. Do you see 42 in the shell?

Answer: _____

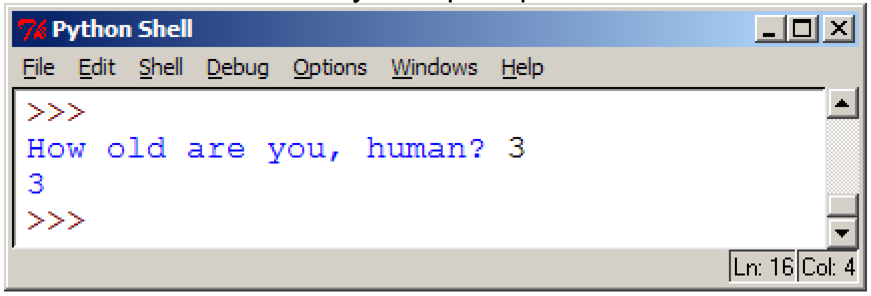
Integer Value from Keyboard

Using IDLE write the following program:



```
x = input("How old are you, human? ")
print x
```

Run it and enter 3 when you're prompted:



```
>>>
How old are you, human? 3
3
>>>
```

The above showed you how to give a variable `x` the value of 3 entered from the keyboard.

Run the program again and enter your age this time.

Do you see **"How old are you, human?"** in the program? Remember strings?

The statement

```
    x = input("How old are you, human? ")
```

basically does the following:

- Python prints the string `"How old are you, human? "`
- Python waits for an integer value from the keyboard
- Once you entered an integer and press the enter key, the integer value you typed is given to `x`.

input is an example of a **function**. Don't worry about the concept of a function for the time being. We'll come back to that soon.

Exercise. What if you execute this program instead:

```
    y = input("")
    print y
```

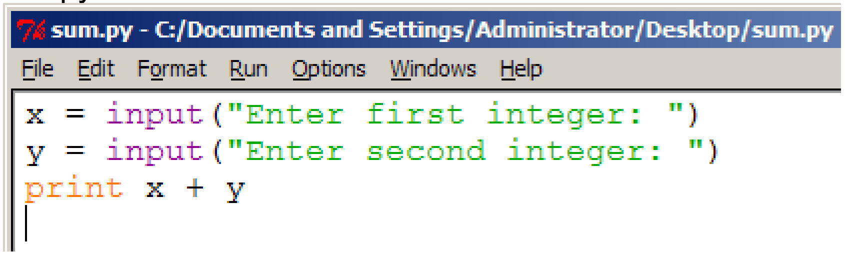
Answer: _____

Exercise. Now what about this:

```
    y = input()
    print y
```

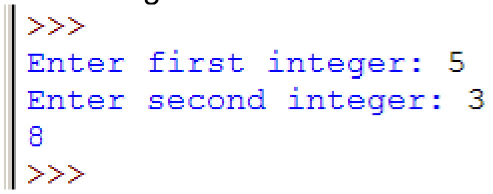
Answer: _____

Exercise. Write the following program. Save the program as sum.py.


sum.py - C:/Documents and Settings/Administrator/Desktop/sum.py
File Edit Format Run Options Windows Help

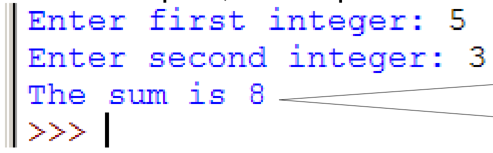
```
x = input("Enter first integer: ")
y = input("Enter second integer: ")
print x + y
```

Run the program. Enter 5 for `x` and 3 for `y` and you should get the following in the shell:



```
>>>
Enter first integer: 5
Enter second integer: 3
8
>>>
```

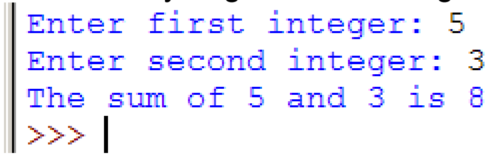
Understand the program. Now change the program so that with the same inputs, the output looks like this:



```
Enter first integer: 5
Enter second integer: 3
The sum is 8
>>> |
```

HINT: It looks like one line of text, but you really want to print **two** things: You want to print the string “The sum is ” and the value of x + y. Remember that you can print several things at the same time.

Exercise. Modify the above program so that with the same inputs as above you get the following output:

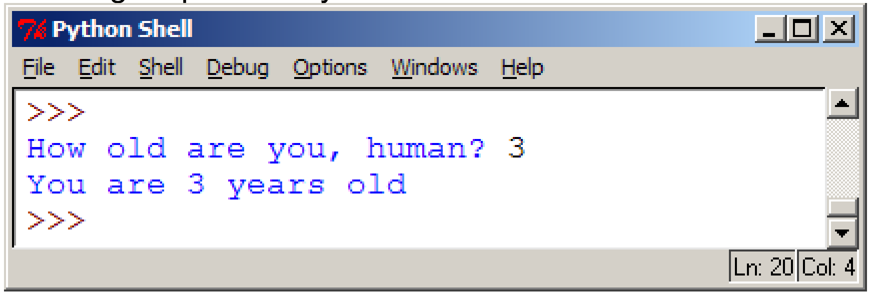


```
Enter first integer: 5
Enter second integer: 3
The sum of 5 and 3 is 8
>>> |
```

HINT: You are printing **6 things!!!** “The sum of “, the value of x, “ and “, the value of y, “is “, and the value of x + y and when you enter 6 and 7, you get

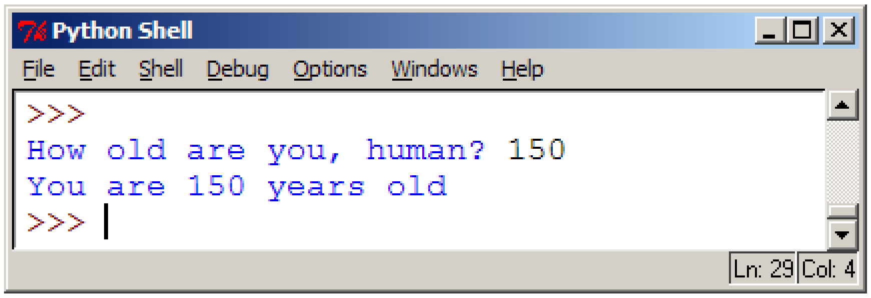
```
Enter first integer: 6
Enter second integer: 7
The sum of 6 and 7 is 13
>>> 
```

Exercise. Write a program that when executed has the following output when you enter 3:



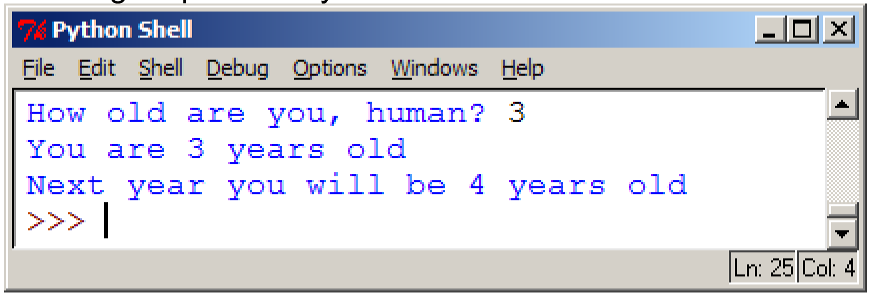
```
>>>
How old are you, human? 3
You are 3 years old
>>>
```

Now if I run the program I get the following when you enter 150:



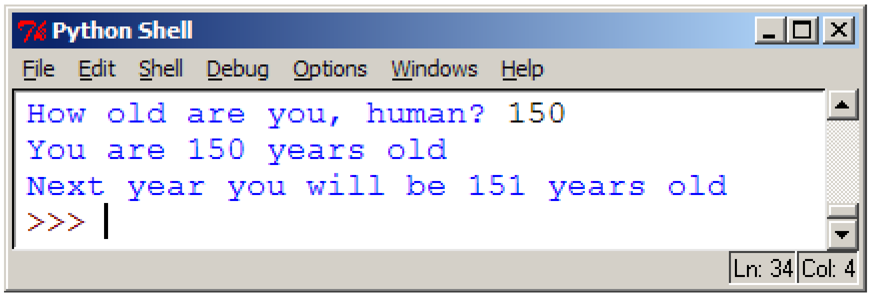
```
>>>
How old are you, human? 150
You are 150 years old
>>> 
```

Exercise. Write a program that when executed has the following output when you enter 3:



```
How old are you, human? 3
You are 3 years old
Next year you will be 4 years old
>>> 
```

And if you run this again and enter 150, you get this:



```
How old are you, human? 150
You are 150 years old
Next year you will be 151 years old
>>> 
```

Variable names

You can use a-z, A-Z, or _ (the underscore) to create variables. So the following is actually a valid Python statement:

```
asdpfiaupdf = 42
```

Now ask yourself, which of the following programs is easier to understand? Here's the first version:

```
sdkfjsldf = input()
print "Per year: ", sdkjsldf / 12
```

and here's the second

```
annual_salary = input()
print "Per year: ", annual_salary / 12
```

Do I need to say more? If you're putting the annual salary of the user of your program into a variable, then it's better to use a variable name close to "annual salary" such as `annual_salary`.

Here are some examples of variable names which are easy to read:

```
number_of_lives
energy_level
```

Exercise. Try to run this program

```
annual salary = input()
print "Per year: ", annual salary / 12
```

What's wrong? What's the point of this "program"?

Answer: _____

Now for the above, I use an underscore (i.e., "_") so that it's easier to see the two words "annual" and "salary" in the variable name `annual_salary`. Another way to do that would be this:

```
annualSalary = input()
```

```
print "Per year: ", annualSalary / 12
```

See that? I used uppercase for the first letter in the words in a variable name except for the first word. Here are some other examples of variable names using this style:

```
framesPerSecond = 60
mainWindowHeight = 480
```

What else can you use in a variable name besides a, ..., z, A, ..., Z and the underscore? You can also use digits, i.e., 0, ..., 9. However 0, ..., 9 **cannot** appear as the **first** thing (i.e. character) in a name. So the following variable names are OK:

```
alien0_energy = 1000
alien1_energy = 2000
```

But this is BAD:

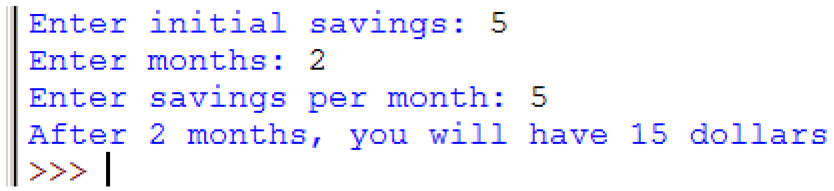
```
1st_place_winner_prize = 100000
```

because the first character in this variable name is `1`.

Exercise. You want to compute how much you will have in your savings accounts after half a year if you start with $100 and if you save $5 per month.

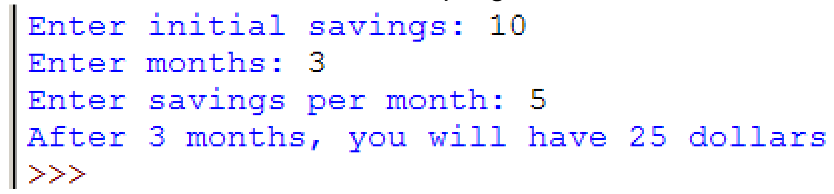
- Create a variable called `savings` and give it a value of 100.
- Create a variable called `savings_per_month` and give it a value of 5.
- Create a variable called `months` and give it a value of 6.
- Now set the variable `savings` to the new value iof the savings after the given time period. This is an expression involving the variable `savings`, `savings_per_month`, and `months`.
- Print the value of the variable `savings`.

Exercise. Now redo the above exercise prompting the user for all the values instead of setting it in the program.



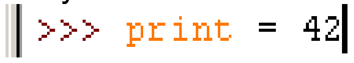
```
Enter initial savings: 5
Enter months: 2
Enter savings per month: 5
After 2 months, you will have 15 dollars
>>> 
```

Here's another execution of the program



```
Enter initial savings: 10
Enter months: 3
Enter savings per month: 5
After 3 months, you will have 25 dollars
>>>
```

OK. One last rule. You cannot use a word that Python has already reserved. For instance try this:



```
>>> print = 42
```

It won't work because `print` has a special meaning to Python. Wouldn't you be confused if every male student in your math class is John??? Or if every female student in your class is Mary? So why confuse Python?

A word already reserved by Python to have a special meaning such as `print` is called a **reserved word** or a **keyword**. You **cannot** use a reserved word for a variable name.

SUMMARY (DIY)

A variable is like the name of a box. You can put values into the box.

Something like

```
    xyz = 1
```

will either

- Put `1` into the box named `xyz` if Python can find such a box

or

- If Python cannot find a box named `xyz`, Python will create a box named `xyz` and put `1` into that box.

If you have something like

```
    xyz = a + b * c
```

then Python will first evaluate the right hand side to get a value and then give it to `xyz`.

A variable name can be created using a-z, A-Z, 0-9, and the underscore but 0-9 cannot appear as the first character in the variable name. You cannot use a keyword for a variable name.

You can also give a variable an integer value entered by the user like this:

```
    xyz = input("Please enter an integer: ")
```

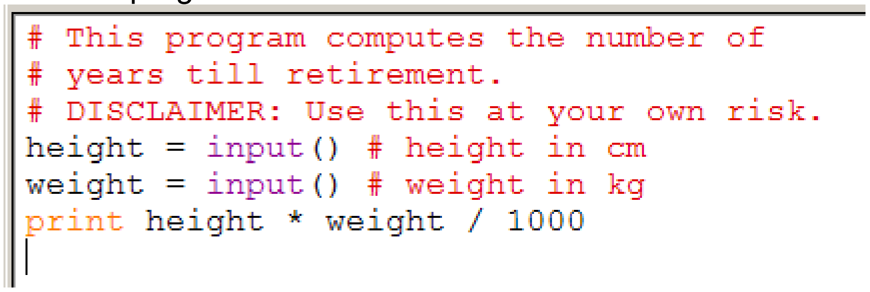
In this case Python will print `"Please enter an integer: "`, wait for an integer value, and give the integer value to `xyz.`

Comments

Good variable names make a program more readable.

Another way to make your program readable is to explain what you're trying to do.

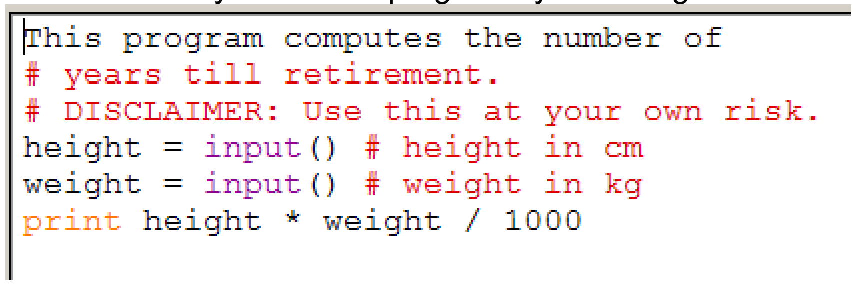
Here's a program. Run it.



```
# This program computes the number of
# years till retirement.
# DISCLAIMER: Use this at your own risk.
height = input() # height in cm
weight = input() # weight in kg
print height * weight / 1000
```

Anything from # to the end of the line is not executed by Python. So you can put (hopefully!) meaningful explanation in your program if you start with #.

Exercise. Modify the above program by removing the first #:



```
This program computes the number of
# years till retirement.
# DISCLAIMER: Use this at your own risk.
height = input() # height in cm
weight = input() # weight in kg
print height * weight / 1000
```

What happens when you run it?
Answer: _____

Bouncing Alien – Integers, Strings, Comments and Variables

Armed with all your knowledge, open bouncing_alien.py with IDLE. The code is on the next page:

```
import sys, pygame

# Initialize Pygame
pygame.init()

# Set surface to 680x480
WIDTH = 640
HEIGHT = 480
SIZE = (WIDTH, HEIGHT)
surface = pygame.display.set_mode(SIZE)
BLACK = (0, 0, 0)

# Set speed of alien to [1,2]
XSPEED = 1
YSPEED = 2
speed = [XSPEED, YSPEED]

# Load alien image and get image rect
alien = pygame.image.load("GalaxianAquaAlien.gif")
alienrect = alien.get_rect()

# Load sound
tag = pygame.mixer.Sound("ChatTag.wav")

while 1:

    # Exit if window is closed
    for event in pygame.event.get():
        if event.type == pygame.QUIT: sys.exit()

    # Move alien
    alienrect = alienrect.move(speed)

    # Deflect along left and right walls
    if alienrect.left < 0 or alienrect.right > WIDTH:
        speed[0] = -speed[0]
        tag.play()

    # Deflect along top and bottom walls
    if alienrect.top < 0 or alienrect.bottom > HEIGHT:
        speed[1] = -speed[1]
        tag.play()

    # Draw surface
    surface.fill(BLACK)
    surface.blit(alien, alienrect)
    pygame.display.flip()
```

Exercise. Identify all the **comments**. Remember: comments help you understand the program. Try to follow the comments and guess what each part of the program is trying to accomplish.

Exercise. There are four integer variables. What are their names and values?

Answer:

WIDTH has value 640 (I'm giving you the first)

____

____

____

____

Exercise. Try to guess what the variables do. First note that the **names of the variables** will be helpful. You might want to change the values of these variables, save the program (using File > Save), and then run the program and see what the change does to the program. For the game, you run the program by double-clicking on the program's icon. Do not run it from IDLE.

HINT: Change WIDTH to 200 and run the program. Change it to 900 and run the program.

Answer:

WIDTH is

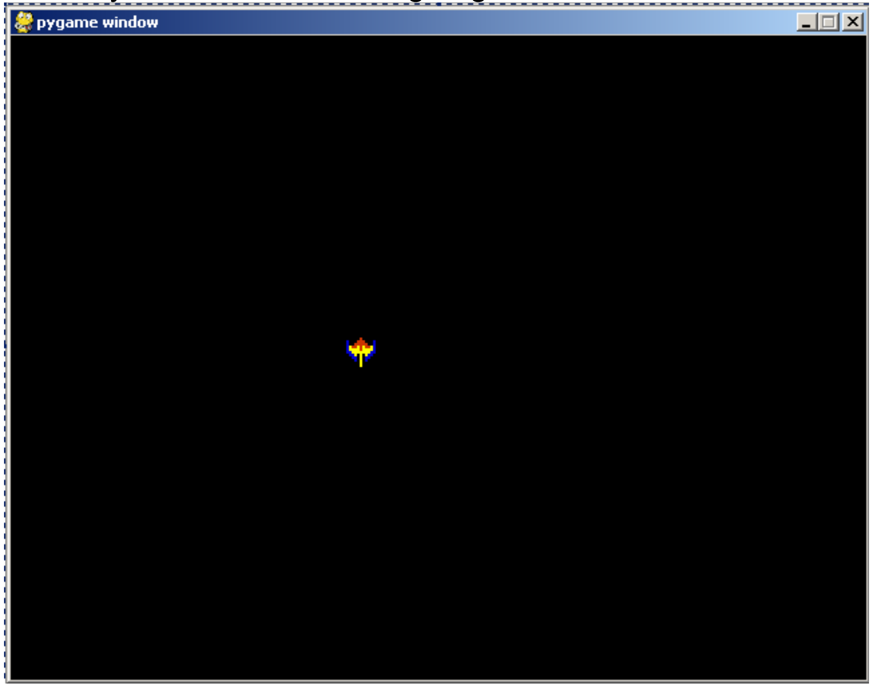
____

____

____

____

Exercise. There are two strings in the program. Can you find them? What do you think they are for? [Hint: Look at the files in the bouncing_alien folder.] Can you change your program so that you have the following bug instead?



Hotkeys and Shortcuts (DIY)

Be a pro. Avoid using the mouse whenever you can by using the keyboard. There are many shortcuts or hotkeys that will make you more efficient.

You already know the following hotkeys in IDLE

Ctrl-N	Open new window (for writing a new program)
F5	Run program

The following are some hotkeys that you can use in Windows:

Ctrl-Esc	Press the “Start” button
Alt-F4	Close a window
Alt-Tab	Switch window

There's a ton of hotkeys/shortcuts. You can use the web to search for more. Don't overload yourself by trying to memorize all of them at once.

Summary

What have you learned? A lot!!!

Topics on programming in general:

- Integer values
- Strings
- Operators on integer values
- Operator precedence
- Variables with integer values
- Comments
- Integer inputs from the keyboard

Python specific knowledge:

- Running Python statements in the IDLE shell.
- Writing and saving Python programs using IDLE.
- Running Python programs in IDLE or double-clicking on a Python program icon.
- Opening a Python program in IDLE for modification.

Extra DIY

Practice makes perfect!

Go over every single page of this set of notes on your own ASAP.

There is a Python book: “Thinking like a computer scientist: Learning with Python”. It's FREE. You can find it on the web. Just google for it. Download the pdf document of the book. (You can also buy a copy if you like). Read Chapter 1, and if possible Chapter 2, before the next meeting.

Looking Ahead

So far everything is still pretty ... **boring**. (Honestly, it is. But you still have to learn the basics before doing exciting stuff.) For instance the program is not smart. Other than doing math, it can't really think. For instance you definitely want to know how to do tell Python something like:

```
if alien is killed increase score by 10
```

You know that `increase score by 10` is the same as `score = score + 10`. You only want Python to execute this statement when something happens, i.e., when an alien is killed. Python can actually understand something like this:

```
if alien is killed:
    score = score + 10
```

The “`alien is killed`” part is not quite correctly. But Python does understand the word `if`. The Python `if` statement looks like this:

```
if [some condition]:
    [some statement]
```

Exercise. Open up bouncing_alien.py. Look for all the if statements. Try to guess what the if statements try to do. Sorry I can't give away more! You have to wait. :)