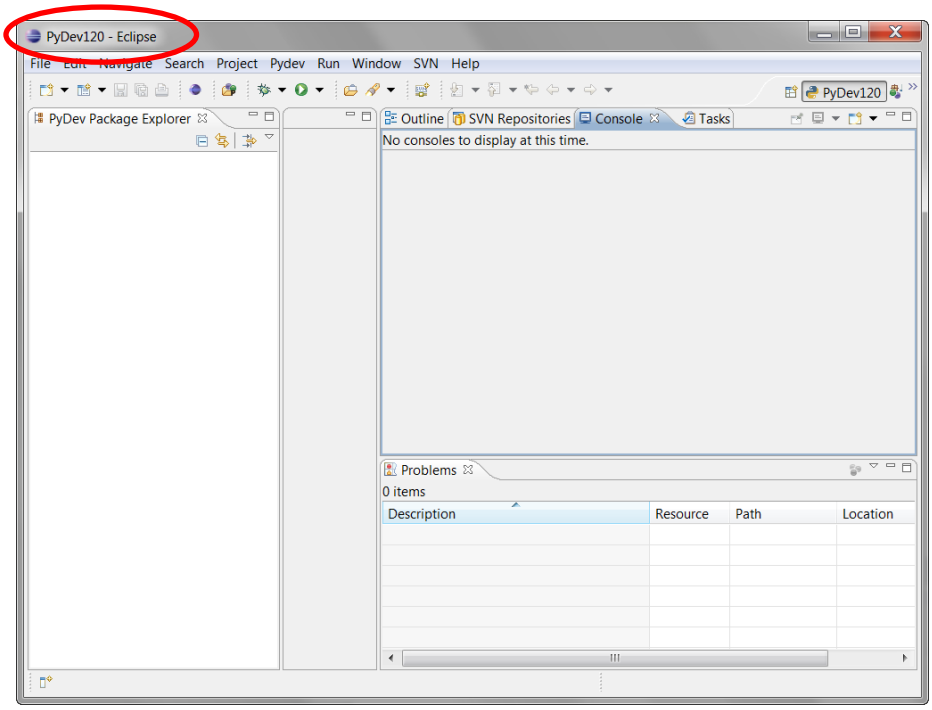
The PyDev Console, by Pictures

Here's an introduction to the ***PyDev Console***, by pictures. It assumes that you are in Eclipse and in the PyDev120 (or PyDev) perspective. If not, your mileage will vary.

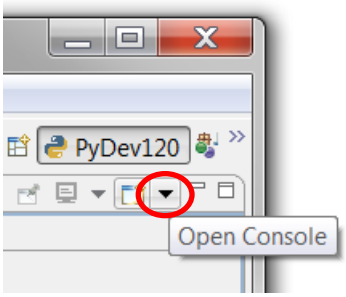
Follow the pictures (and associated instructions), one by one. ***Turn to a neighbor or an assistant quickly whenever you get a bit lost*** – most of this is much easier to show than to explain.

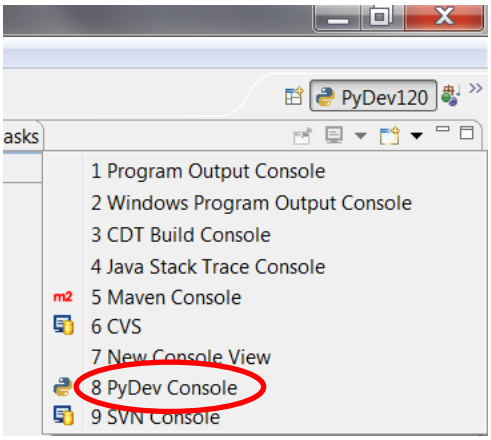


The PyDev Console lets you type and run Python commands (aka statements), one at a time.

1. Your window should look like this, except perhaps for the window sizes.
   - I kept my window small only to make it fit better in these instructions. ***You'll want your Eclipse window maximized.***

2. ***Open a PyDev Console***, as shown in the ***next several pictures*** (all of which are blow-ups of parts of the above picture, near the upper-right corner).





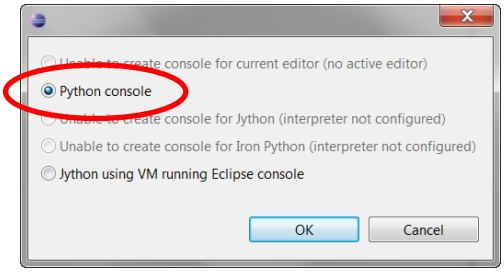
Start with the pull-down arrow that brings up an Open Console pop-up if you hover. It's tricky to locate, ask someone to show you).

From the pull-down menu, you select

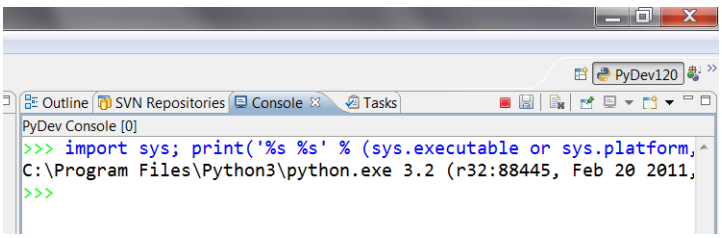
PyDev Console

and then in the dialog box that pops up:

Python console



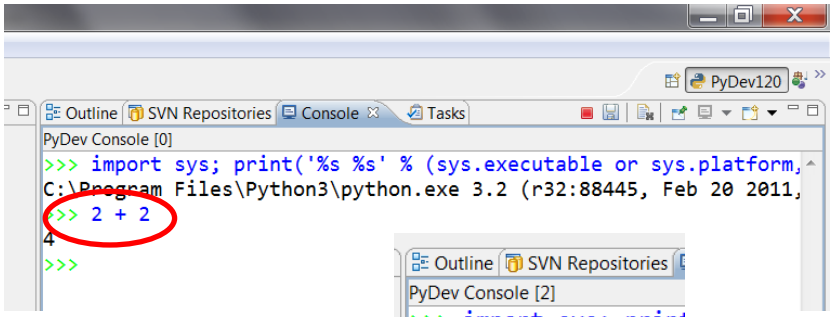
The result should be something like the picture below.

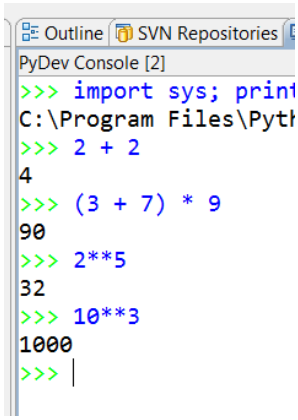


3. This window, with the **green triple arrows >>>**, is a ***PyDev Console***. Use it whenever you want to try out a single command (aka statement).

   For now, try typing some arithmetic (the picture to the right shows **2 + 2**) at the triple green arrows and press ***Enter***.

   Once you do so – congratulations! You have successfully executed your first Python command (aka statement)





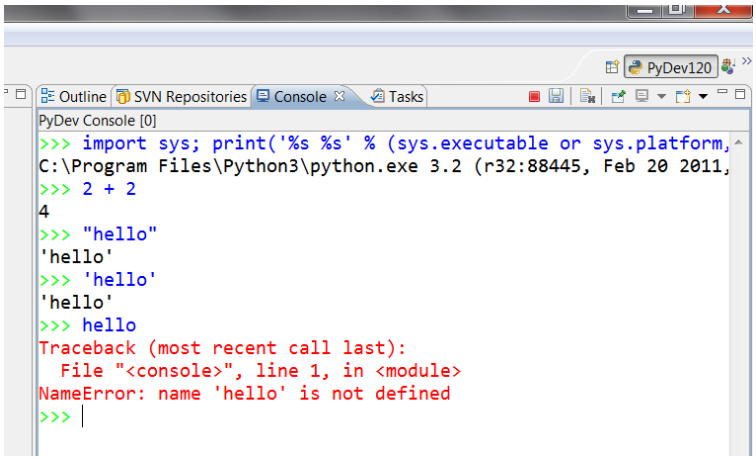
4. ***Do a few more arithmetic expressions***, until you feel comfortable with arithmetic in the PyDev Console. Choose your own numbers, including some with decimal points (which we call *floating point numbers*, or simply *floats*); the example to the right is just a guide. Can you make a very, very large number show up as the result of a calculation?

5. Now let's try **strings** – *sequences of characters*. Try typing some word (whatever word you want) in double quotes, in single quotes and with no quotes. You should see something similar to the picture to the right.

   Pause for a moment to think briefly about what the last line of that error message might mean:

```
NameError: name 'hello' is not defined
```

   What's a name? Can you guess?



6. We'll talk lots more about ***names*** (aka ***variables***) in the next few sessions. For now, take a quick look at the picture to the right to see how you can define names to have ***values***: the name ***`hello`*** is given the *integer* value 54 and the name ***`greeting`*** is given the *string* value `'hello'`.

   If you have time, ***try some string "arithmetic"*** as shown to the right, with your own strings and numbers. Don't hesitate to ask an assistant or a neighbor questions. This is just play time!

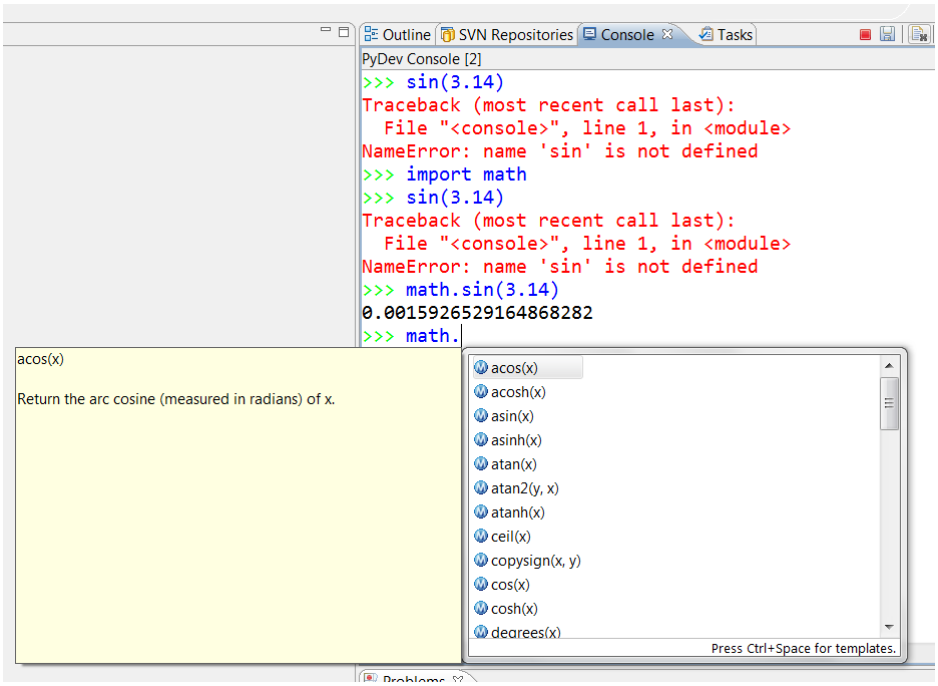
7. Last concept for now: You can call ***functions*** somewhat like you do in math class. Try statements like the ones to the right, including the two that yield error messages.

   Next session, we'll talk more about the need to ***import*** the **math** module and why we need to write **`math.sin`** instead of just **`sin`**.

   Do try to notice that when you type

   `math.`

   (note the *DOT*) and then *PAUSE for a second or two*, a pop-up window shows you all the functions in the math module. Cool, no?!



8. Finally, some functions are *built-in*. The ***abs*** function shown to the right is one such.

   You don't have to import the built-in's or precede those functions with the name ***builtins*** and a dot, but you can if you wish. So really, built-ins are much like functions in other modules.

```
>>>
>>>
>>> math.abs(-4)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
AttributeError: 'module' object has no attribute 'abs'
>>> abs(-4)
4
>>> builtins.abs(-4)
4
>>>
```