

Introduction to Computer Science

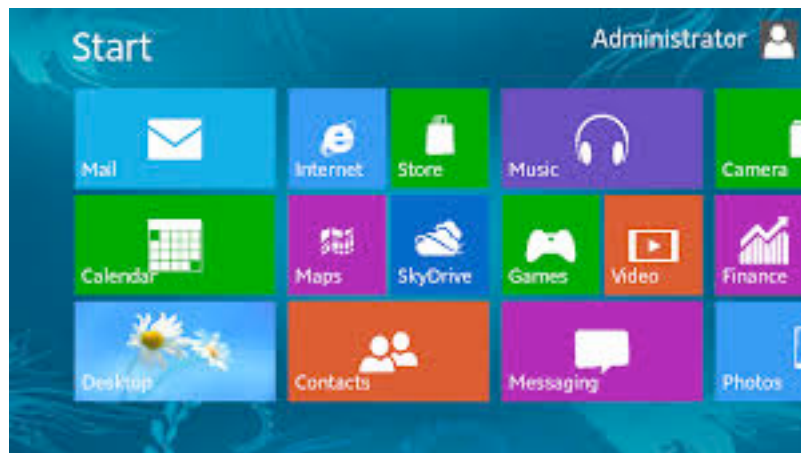
Python Basics

鄒年棣 (Nien-Ti Tsou)

Computing Systems

- The operating system (OS) serves as the **manager** of the computer system as a whole containing a group of programs called utilities that allow you to perform basic tasks.

Windows



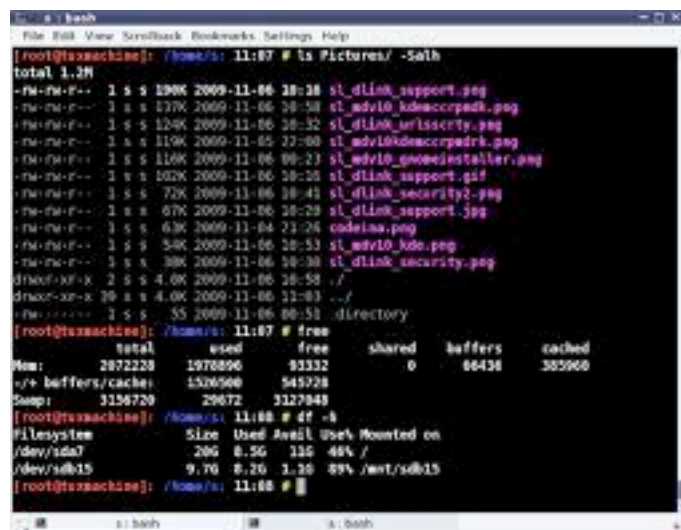
OS X



Utilities



Linux

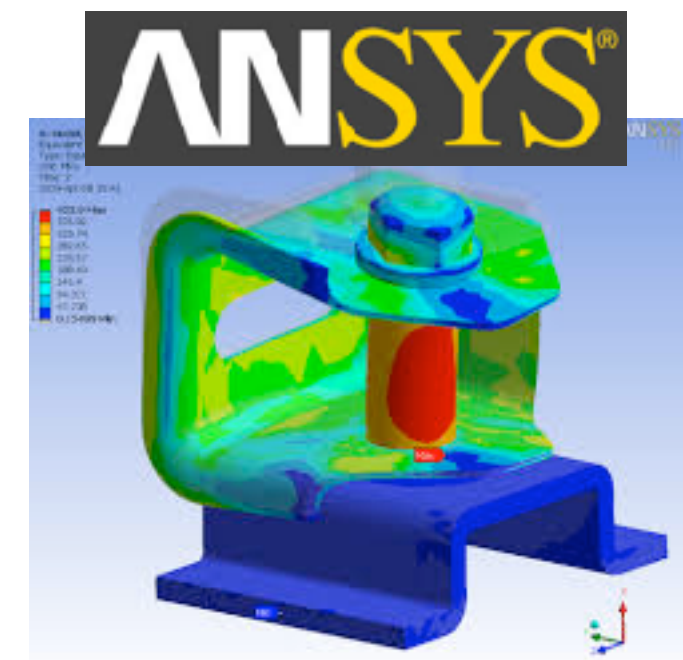
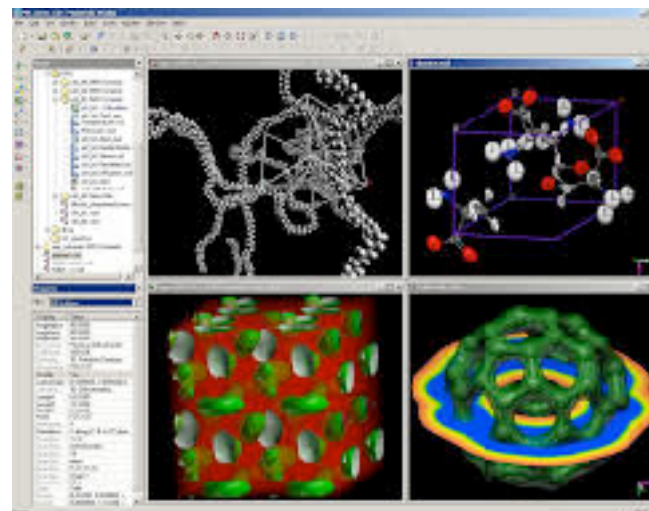


ubuntu (Linux)



Applications

- Software tools are commercial programs that have been written to solve specific problems.
- word processors enable you to enter and format text and graphics.
- spreadsheets let you work with data displayed in a grid of rows and columns.
- Computer-aided design (CAD) packages let you define computer models of real-world objects



Useful applications

- Cloud storage
 - Dropbox, Google drive



- Reference manager
 - EndNote, Mendeley, Zotero



- BBS browser
 - PCMAN, Nally

特別推薦：PTT Python版



Programming Languages

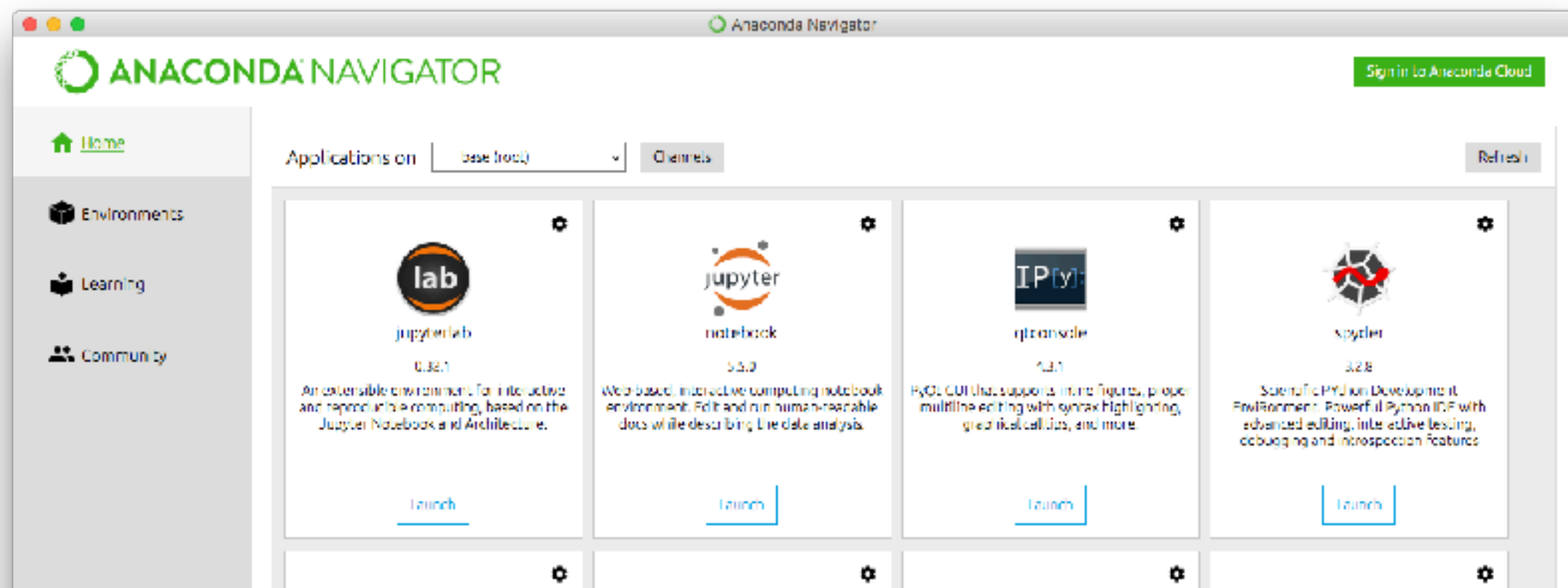
- A Programming Language is a tool a programmer uses to express the **logic** for a computer to implement.
 - defined by its grammar (**syntax**) and its vocabulary.
- Three attributes of a computer language:
 - the scope of the **logic** expressed in each line of code (the power of the language),
 - the clarity of each line of code from the **human viewpoint**, and
 - **portability** between different types of processor

Python

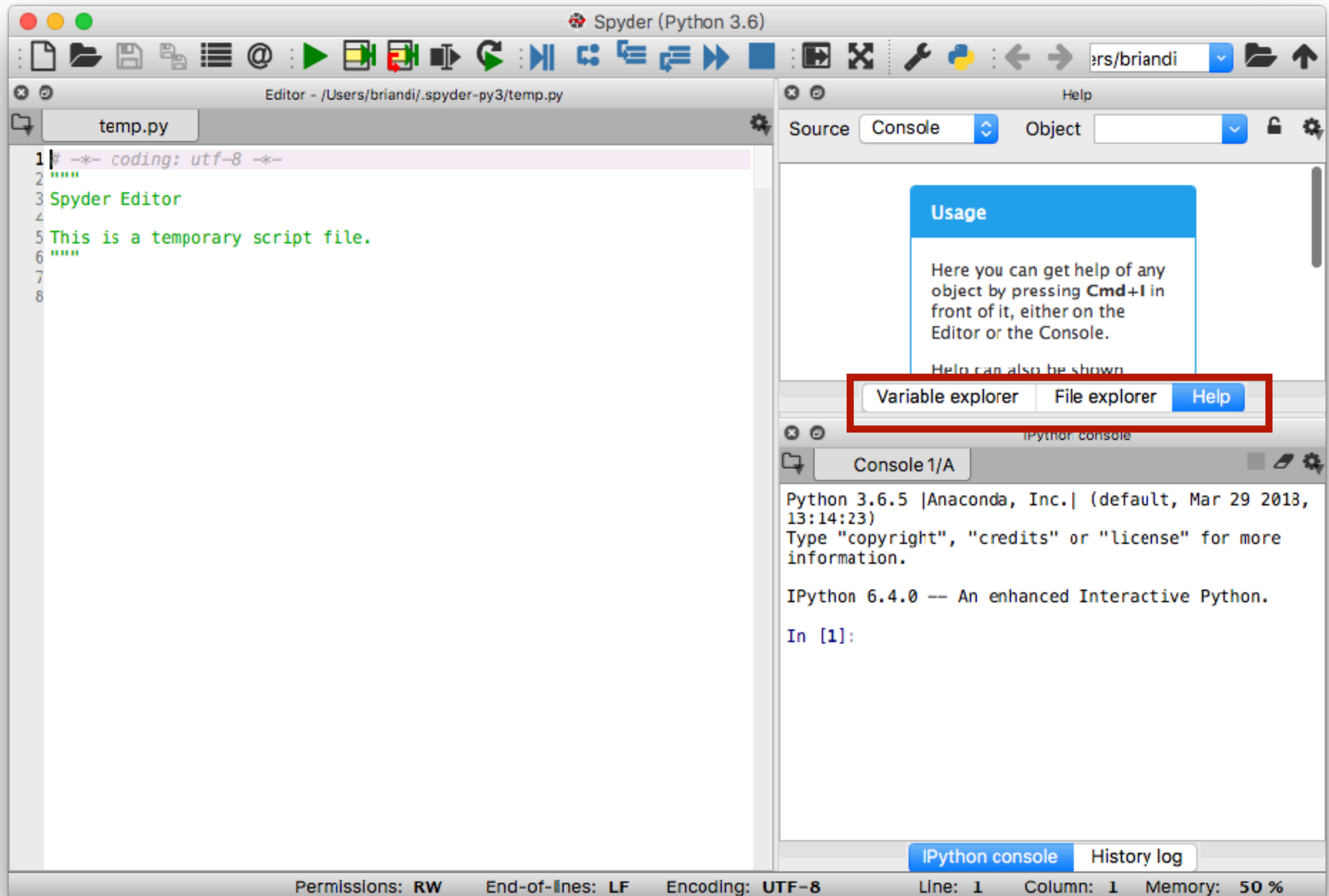
- Free and open source
- A **scripting language**, meaning that it is interpreted.
- A modern language (**object oriented**, exception handling, etc.)
- Concise, **easy** to read and quick to learn.
- Full of **freely available libraries**, in particular scientific one.
- Widely used in industrial applications.
- Java, C++ : Object oriented, compiled languages. More verbose and low level compared to Python. Few scientific libraries.
- MATLAB: a tool for matrix computation that evolved for scientific computing. The scientific library is huge. The language features are not as developed as those of Python. Neither free nor open source.

Anaconda

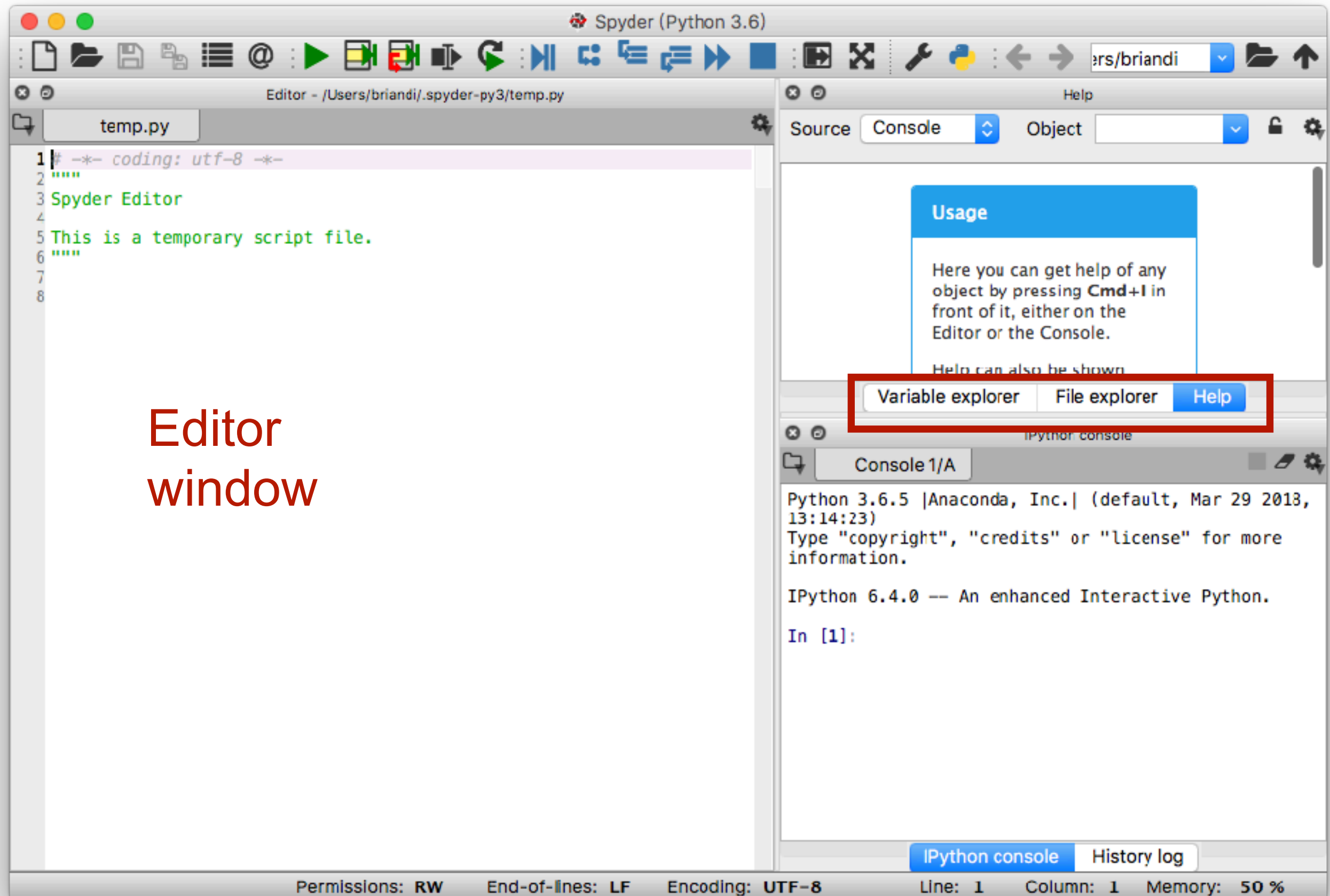
- Create your personal Python environment.
- With a **virtual environment**, you are free to change language versions and install packages without the unintended side-effects. If the worst happens and you screw things up totally, just delete the Anaconda directory and start again.
- Running the Anaconda installer will install Python, a Python development environment and editor (**Spyder**), the shell **IPython**, and the most important **packages** for numerical computations, for example **SciPy**, **NumPy**, and **matplotlib**.



Editor: Spyder

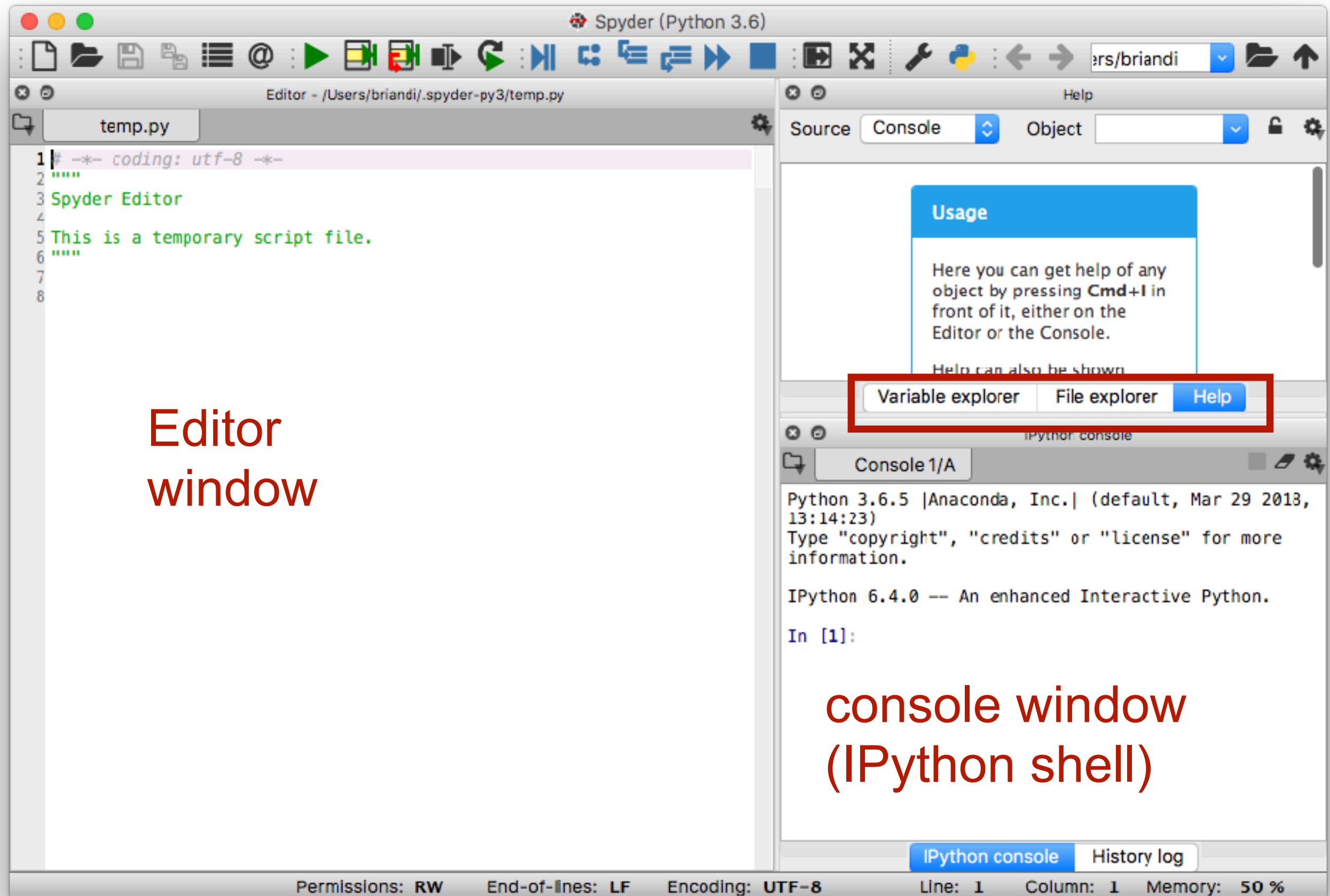


Editor: Spyder



Editor
window

Editor: Spyder



Editor
window

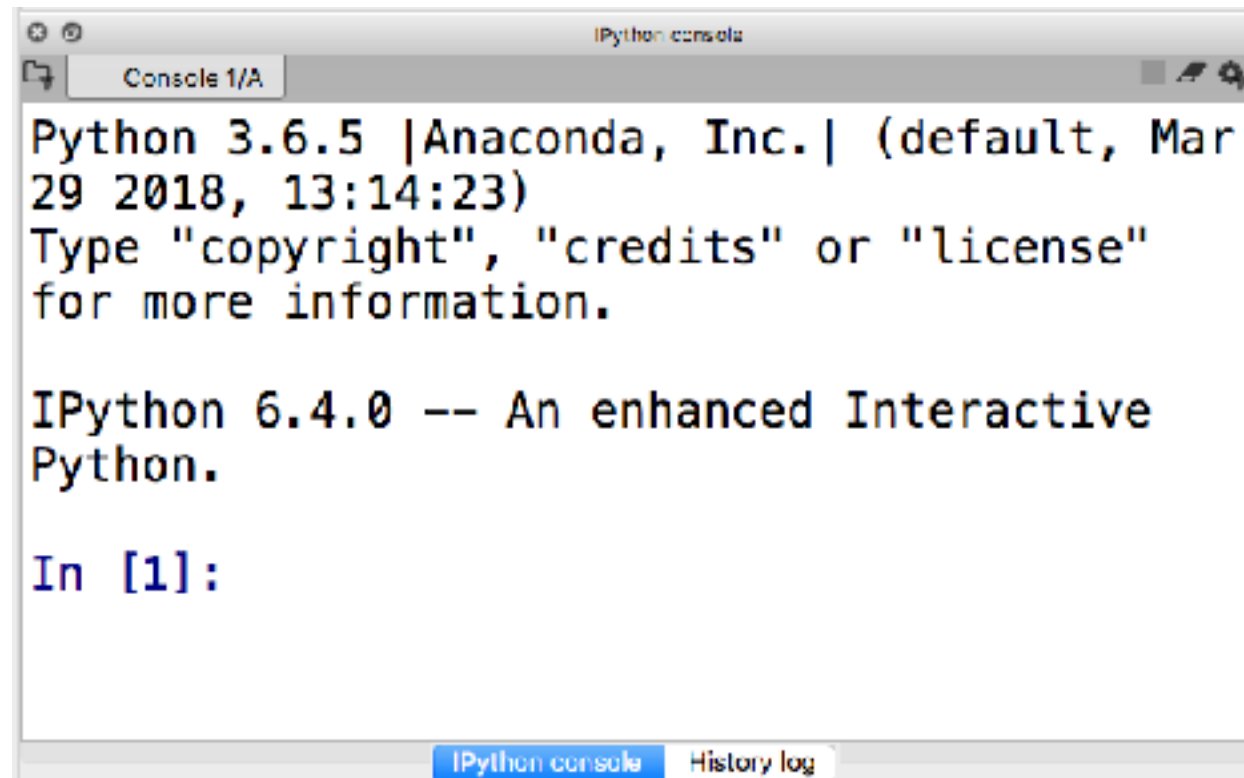
console window
(IPython shell)

Python Shell

- Shell is a user interface for access to an operating system's services. In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI). It is the outermost layer around the operating system kernel.
- The Python shell is good but not optimal for interactive scripting. We therefore recommend using IPython instead.
- IPython (Interactive Python) is a command shell for **interactive** computing, that offers introspection, rich media, shell syntax, tab completion, and history. IPython provides the following features

Running the Python interactive shell

IPython



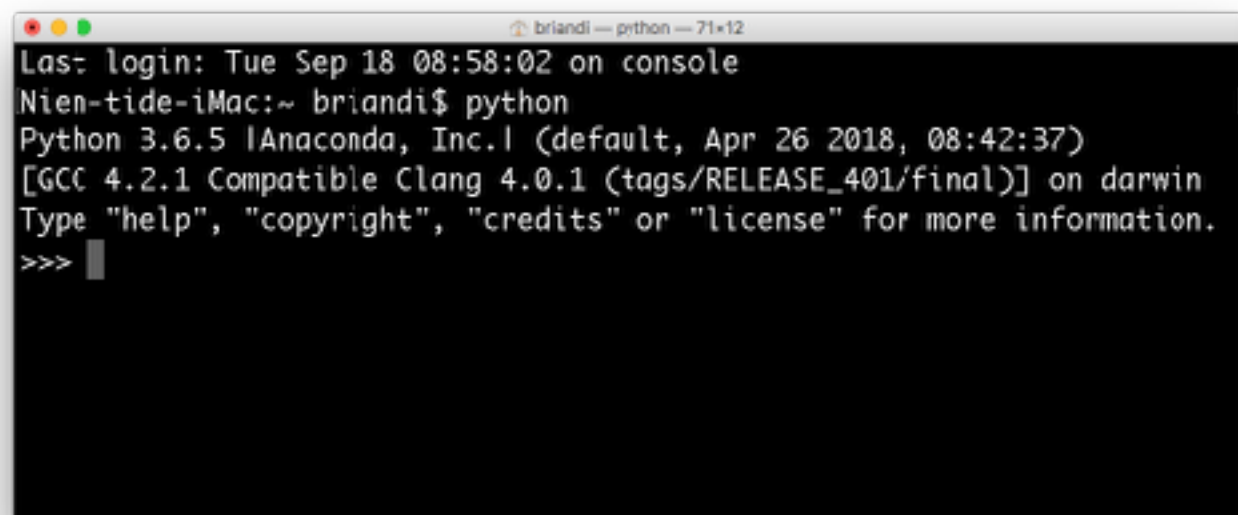
```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:14:23)
Type "copyright", "credits" or "license"
for more information.

IPython 6.4.0 -- An enhanced Interactive
Python.

In [1]:
```

- `In [1]:` or `>>>` tell you that Python is waiting for you to type something.
- you give a command, and the shell tries to execute it right away, then awaits another.

Python shell



```
Last login: Tue Sep 18 08:58:02 on console
Nien-tide-iMac:~ briandi$ python
Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
In [1]: 2 + 4
Out[1]: 6

In [2]: 2**1024
Out[2]:
1797693134862315907729305190789024733617976
9789423065727343008115773267580550096313270
8477322407536021120113879871393357658789768
8144166224928474306394741243777678934248654
8527630221960124609411945308295208500576883
8150682342462881473913110540827237163750510
684586298239947245938479716304835356785242
24137216
```



Try to do it in Java, C++, or C#. It won't work, unless you use special libraries to handle such big numbers.

Some Notes

- Comments: If a line in a program contains the symbol #, everything following on the same line is considered as a comment.

```
# This is a comment  
a = 3 # This is a comment, too
```

- Line joining: A backslash \ at the end of the line marks the next line as a continuation line.

```
a = \  
3
```


Some Notes

- Comments: If a line in a program contains the symbol #, everything following on the same line is considered as a comment.

```
# This is a comment  
a = 3 # This is a comment, too
```

- Line joining: A backslash \ at the end of the line marks the next line as a continuation line.

```
a = \  
3
```

請不要作這麼無聊的事

Variables (變數)

- Variables (變數): A **reference** (引用) to an **object** (物件). An object may have several references. Use the assignment operator = to **assign** (指定) a value to a variable

```
# assign 4 to the variable x  
x=4
```

- You defined a **pointer** named x that points to some other bucket containing the value 4.
- Python is **dynamically typed**: variable names can point to objects of **any type**.

```
x = 4 # x is an integer  
x = 'hello' # now x is a string
```

Are they correct?

$$2+3 = x$$

$$x-y = z$$

Are they correct?

$$2+3 = x$$

$$x-y = z$$

Variable explorer

```
In [3]: a = 3
```

```
In [4]: x = 4
```

```
In [5]: x = 'hello'
```



Variable explorer window showing the current state of variables. The table lists variables 'a' and 'x', both of type 'int' with size 1. The values are 3 and 4 respectively.

Name	Type	Size	Value
a	int	1	3
x	int	1	4



Variable explorer window showing the current state of variables after the third input. The table lists variables 'a' and 'x'. Variable 'a' is of type 'int' with size 1 and value 3. Variable 'x' is of type 'str' with size 1 and value 'hello'.

Name	Type	Size	Value
a	int	1	3
x	str	1	hello

Basic types (資料型態)

- Numbers: an integer, a real number, or a complex number

```
a = 2 ** (2 + 2) # 16  
b = 1j ** 2 # (-1+0j)
```

```
>>> b=1*j  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'j' is not defined
```

- Strings (字串): Sequences of characters, enclosed by simple quote ' ' or double quote " ". Triple quotes for multiple lines

```
Onestr = 'these are double quotes: ".." '  
Morestr = """ Many lines 1~  
          Many lines 2~  
Many lines 3~"""
```

`\n`: New line

```
In [2]: Onestr  
Out[2]: 'these are double quotes: ".." '  
  
In [3]: Morestr  
Out[3]: ' Many lines 1~\n      Many lines  
2~\nMany lines 3~'
```

Basic types (資料型態)

- Numbers: an integer, a real number, or a complex number

```
a = 2 ** (2 + 2) # 16  
b = 1j ** 2 # (-1+0j)
```

```
>>> b=1*j  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'j' is not defined
```

- Strings (字串): Sequences of characters, enclosed by simple quote ' ' or double quote " ". Triple quotes for multiple lines

```
Onestr = 'these are double quotes: ".."'  
Morestr = """Many lines 1~  
Many lines 2~  
Many lines 3~"""
```

\n: New line

```
In [2]: Onestr  
Out[2]: 'these are double quotes: "..'  
'  
  
In [3]: Morestr  
Out[3]: ' Many lines 1~\n      Many lines  
2~\nMany lines 3~'
```

Basic types (資料型態)

- Lists: An ordered **list of objects** enclosed by **square brackets**.
- One can access the elements of a list using **zero-based indexes inside square brackets**

```
L1 = [5, 6]
print(L1[0])
print(L1[1])
print(L1[2])
```


```
L2 = ['a', 1, [3, 4]]
print(L2[0])
print(L2[2][0])
print(L2[-1])
print(L2[-2])
```

print (): A built-in function that show the content of the object on the screen

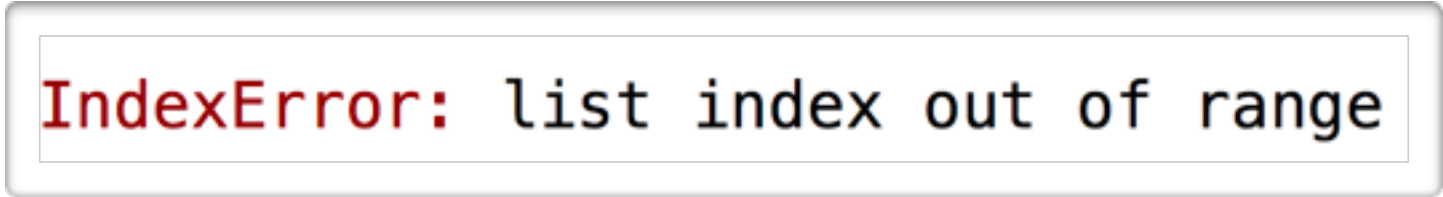
Basic types (資料型態)

- Lists: An ordered **list of objects** enclosed by **square brackets**.
- One can access the elements of a list using **zero-based indexes inside square brackets**

```
L1 = [5, 6]  
print(L1[0])  
print(L1[1])  
print(L1[2])
```



5
6



IndexError: list index out of range

```
L2 = ['a', 1, [3, 4]]  
print(L2[0])  
print(L2[2][0])  
print(L2[-1])  
print(L2[-2])
```

print (): A built-in function that show the content of the object on the screen

Basic types (資料型態)

- Lists: An ordered **list of objects** enclosed by **square brackets**.
- One can access the elements of a list using **zero-based indexes inside square brackets**

```
L1 = [5, 6]  
print(L1[0])  
print(L1[1])  
print(L1[2])
```

```
5  
6
```

```
IndexError: list index out of range
```

```
L2 = ['a', 1, [3, 4]]  
print(L2[0])  
print(L2[2][0])  
print(L2[-1])  
print(L2[-2])
```

```
a  
3  
[3, 4]  
1
```

print (): A built-in function that show the content of the object on the screen

Operations on lists

- The operator + concatenates two lists

```
L1 = [1, 2]
L2 = [3, 4]
L = L1 + L2
```

- Multiplying a list with an integer n will repeat the list with itself n times.

```
L = [1, 2]
3 * L
```

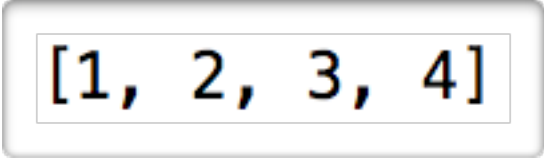
- String has similar behavior:

```
a = 'hello'
a = a+a
print(a)
print(a[-1])
```

Operations on lists

- The operator + concatenates two lists

```
L1 = [1, 2]  
L2 = [3, 4]  
L = L1 + L2
```



```
[1, 2, 3, 4]
```

- Multiplying a list with an integer n will repeat the list with itself n times.

```
L = [1, 2]  
3 * L
```

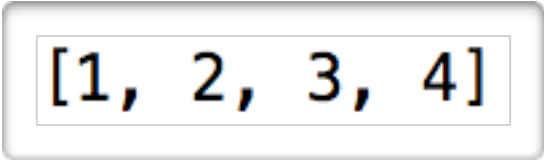
- String has similar behavior:

```
a = 'hello'  
a = a+a  
print(a)  
print(a[-1])
```

Operations on lists

- The operator + concatenates two lists

```
L1 = [1, 2]  
L2 = [3, 4]  
L = L1 + L2
```



```
[1, 2, 3, 4]
```

- Multiplying a list with an integer n will repeat the list with itself n times.

```
L = [1, 2]  
3 * L
```



```
[1, 2, 1, 2, 1, 2]
```

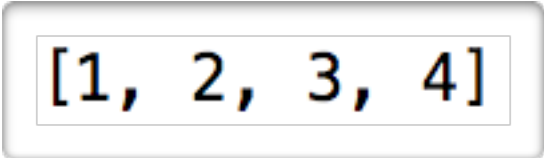
- String has similar behavior:

```
a = 'hello'  
a = a+a  
print(a)  
print(a[-1])
```

Operations on lists

- The operator + concatenates two lists

```
L1 = [1, 2]
L2 = [3, 4]
L = L1 + L2
```



```
[1, 2, 3, 4]
```

- Multiplying a list with an integer n will repeat the list with itself n times.

```
L = [1, 2]
3 * L
```



```
[1, 2, 1, 2, 1, 2]
```

- String has similar behavior:

```
a = 'hello'
a = a+a
print(a)
print(a[-1])
```

```
'hellohello'
```



```
o
```

Some basic list functions

- `list(range(n))` creates a list with `n` elements, starting with **zero**

```
print(list(range(5)))
```

`range(start, stop, step)`: A built-in function generating sequence of numbers

- `len` gives the length of a list:

```
len(['a', 1, 2, 34])
```

- `append` is used to append an element to a list

```
L = ['a', 'b', 'c']  
print(L[-1])  
L.append('d')  
print(L)
```


Some basic list functions

- `list(range(n))` creates a list with `n` elements, starting with **zero**

```
print(list(range(5)))
```

```
[0, 1, 2, 3, 4]
```

`range(start, stop, step)`: A built-in function generating sequence of numbers

- `len` gives the length of a list:

```
len(['a', 1, 2, 34])
```

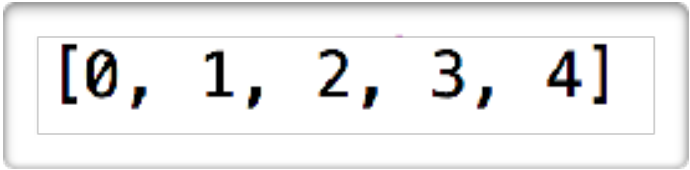
- `append` is used to append an element to a list

```
L = ['a', 'b', 'c']  
print(L[-1])  
L.append('d')  
print(L)
```

Some basic list functions

- `list(range(n))` creates a list with `n` elements, starting with **zero**

```
print(list(range(5)))
```



[0, 1, 2, 3, 4]

`range(start, stop, step)`: A built-in function generating sequence of numbers

- `len` gives the length of a list:

```
len(['a', 1, 2, 34])
```



4

- `append` is used to append an element to a list

```
L = ['a', 'b', 'c']  
print(L[-1])  
L.append('d')  
print(L)
```

Some basic list functions

- `list(range(n))` creates a list with `n` elements, starting with **zero**

```
print(list(range(5)))
```

```
[0, 1, 2, 3, 4]
```

`range(start, stop, step)`: A built-in function generating sequence of numbers

- `len` gives the length of a list:

```
len(['a', 1, 2, 34])
```

```
4
```

- `append` is used to append an element to a list

```
L = ['a', 'b', 'c']  
print(L[-1])  
L.append('d')  
print(L)
```

```
c  
['a', 'b', 'c', 'd']
```

From A to A+!

More detail about variables

- Variables take names that consist of any combination of capital and small letters, the underscore `_`, and digits. Such as `diameter1`, `Height_2`. A variable name must **not start with a digit**, such as `1Variable` (wrong!!).
- Variable names are **case sensitive**.
- Python has some **reserved keywords**, which **cannot** be used as variable names. As shown below:

<code>and</code>	<code>as</code>	<code>assert</code>	<code>break</code>	<code>class</code>	<code>continue</code>	<code>def</code>	<code>del</code>
<code>elif</code>	<code>else</code>	<code>except</code>	<code>exec</code>	<code>False</code>	<code>finally</code>	<code>for</code>	<code>from</code>
<code>global</code>	<code>if</code>	<code>import</code>	<code>in</code>	<code>is</code>	<code>lambda</code>	<code>None</code>	<code>nonlocal</code>
<code>not</code>	<code>or</code>	<code>pass</code>	<code>raise</code>	<code>return</code>	<code>True</code>	<code>try</code>	<code>while</code>
<code>yield</code>							

More detail about variables

- You can create several variables with a multiple assignment statement:

```
a = b = c = 1 # a, b and c get the same value 1
```

- Variables can also be altered after their definition:

```
a = 1
```

```
a = a + 1 # a gets the value 2
```

```
a = 3 * a # a gets the value 6
```

- The last two statements can be written by combining the two operations with an assignment directly by using increment operators:

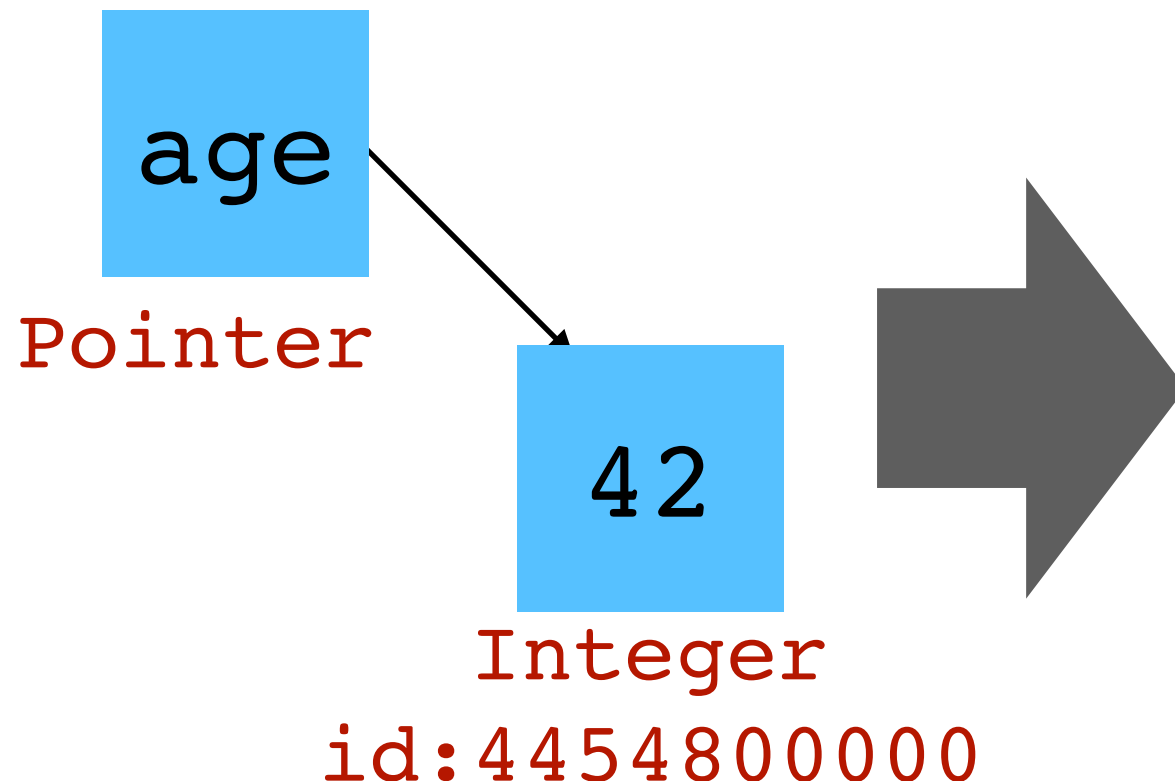
```
a += 1 # same as a = a + 1
```

```
a *= 3 # same as a = 3 * a
```

More detail about variables

- Since Python variables just **point to** various objects, there is **no need to “declare”** the variable, or even require the variable to always point to information of the **same type**.

```
age = 42
print(id(age))
age = 43
print(id(age))
age2 = 42
print(id(age2))
```

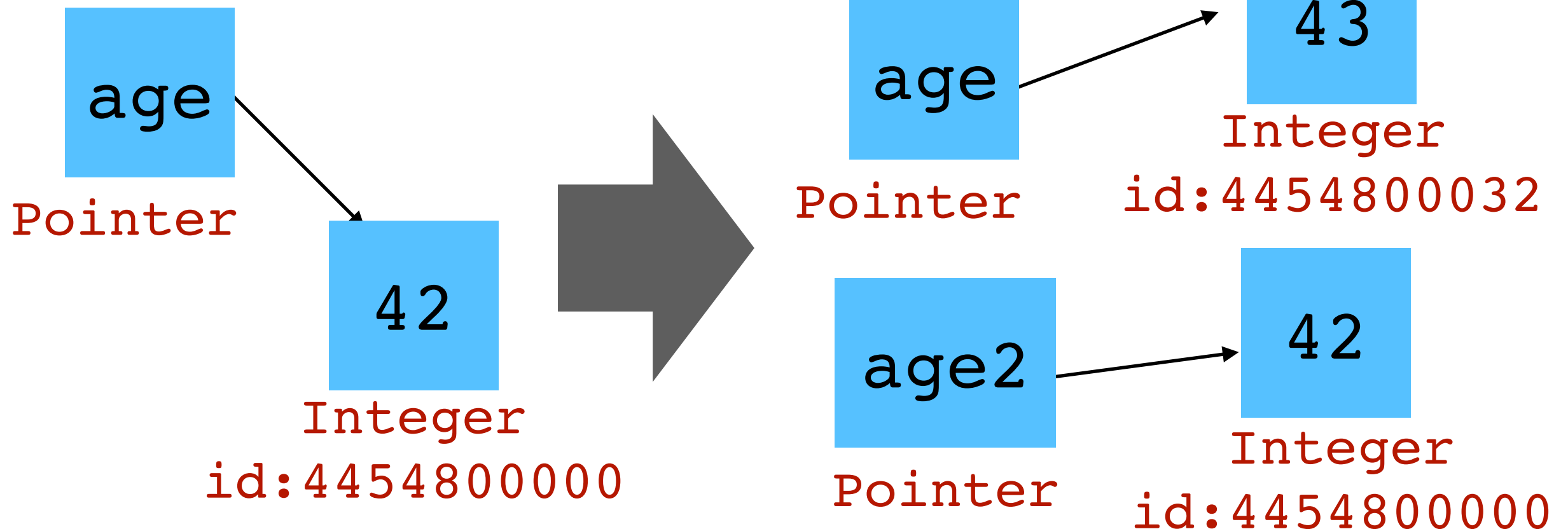


More detail about variables

- Since Python variables just **point to** various objects, there is **no need to “declare”** the variable, or even require the variable to always point to information of the **same type**.

```
age = 42
print(id(age))
age = 43
print(id(age))
age2 = 42
print(id(age2))
```

4454800000
4454800032
4454800000



More detail about variables

- “Variable as pointer”
- If we have two variable names pointing to the same **mutable object**, then **changing one will change the other** as well!

```
x=[1,2,3]
y=x
print(y)
x.append(4) # append 4 to the list pointed by x
print(y)
```

```
x=[1,2,3]
y=[1,2,3]
print(y)
x.append(4) # append 4 to the list pointed by x
print(x)
print(y)
```

More detail about variables

- “Variable as pointer”
- If we have two variable names pointing to the same **mutable object**, then **changing one will change the other** as well!

```
x=[1,2,3]
y=x
print(y)
x.append(4) # append 4 to the list pointed by x
print(y)
```

[1, 2, 3]
[1, 2, 3, 4]

```
x=[1,2,3]
y=[1,2,3]
print(y)
x.append(4) # append 4 to the list pointed by x
print(x)
print(y)
```

More detail about variables

- “Variable as pointer”
- If we have two variable names pointing to the same **mutable object**, then **changing one will change the other** as well!

```
x=[1,2,3]
y=x
print(y)
x.append(4) # append 4 to the list pointed by x
print(y)
```

[1, 2, 3]
[1, 2, 3, 4]

```
x=[1,2,3]
y=[1,2,3]
print(y)
x.append(4) # append 4 to the list pointed by x
print(x)
print(y)
```

[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3]

Mutable and immutable objects

- **Mutable object**: like lists. They **modify the contents** of the original object **rather than creating a new object** to store the result.
- **Immutable object**: like **numbers**, strings, tuples. This means that once they are created, their size and contents **cannot be changed**.

```
TestList = ['a', 'b', 'c']  
print(TestList)  
TestString = 'abc'  
print(TestString)  
TestList[1]='x'  
print(TestList)  
TestString[1] = 'x'
```

Mutable and immutable objects

- **Mutable object**: like lists. They **modify the contents** of the original object **rather than creating a new object** to store the result.
- **Immutable object**: like **numbers**, strings, tuples. This means that once they are created, their size and contents **cannot be changed**.

```
TestList = ['a', 'b', 'c']  
print(TestList)  
TestString = 'abc'  
print(TestString)  
TestList[1]='x'  
print(TestList)  
TestString[1] = 'x'
```

```
['a', 'b', 'c']  
abc  
['a', 'x', 'c']
```

Mutable and immutable objects

- **Mutable object**: like lists. They **modify the contents** of the original object **rather than creating a new object** to store the result.
- **Immutable object**: like **numbers**, strings, tuples. This means that once they are created, their size and contents **cannot be changed**.

```
TestList = ['a', 'b', 'c']  
print(TestList)  
TestString = 'abc'  
print(TestString)  
TestList[1]='x'  
print(TestList)  
TestString[1] = 'x'
```

```
['a', 'b', 'c']  
abc  
['a', 'x', 'c']
```

```
TypeError: 'str' object does  
not support item assignment
```