SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

- ❖ A readable, dynamic, pleasant, flexible, fast and powerful language

UNIT II DATA TYPES, EXPRESSIONS, STATEMENTS

- Python interpreter and interactive mode, debugging; values and types: int, float, boolean, string , and list; **variables,** expressions, statements, tuple assignment, precedence of operators, comments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

Recap

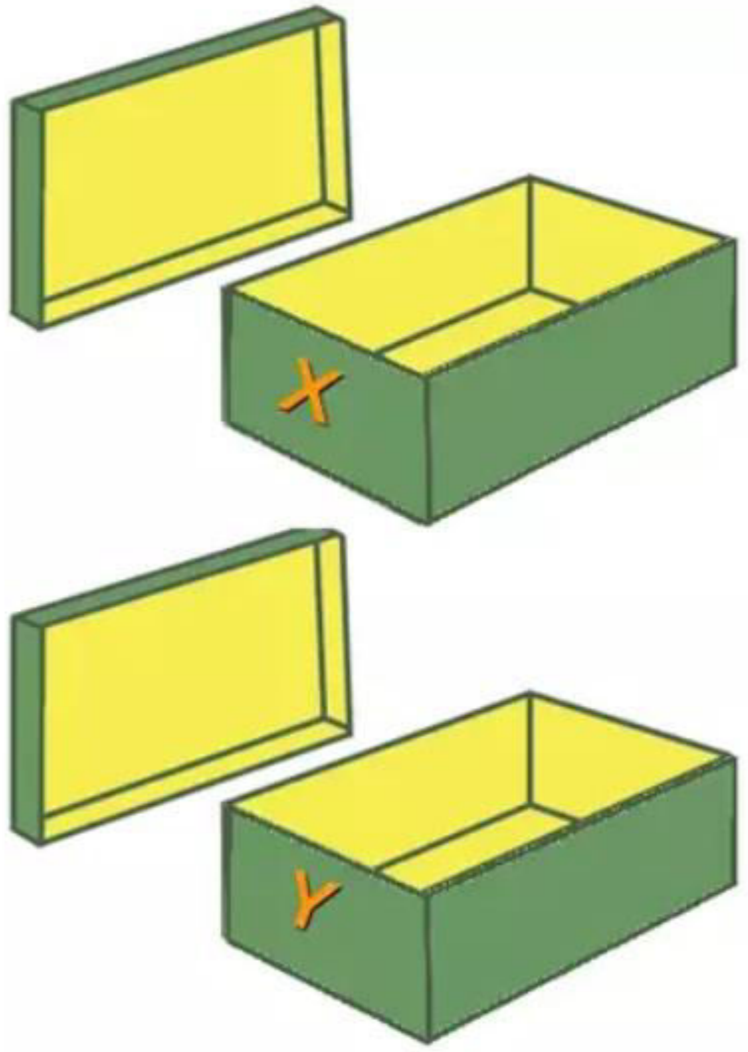
- **Values and Types**
  - List
  - Dictionary
  - Tuple
  - Set

Variables

- As the name implies, a **variable** is something which **can change**.
- A variable is a way of **referring to a memory location** used by a computer program.
- This memory location contains **values, like numbers, text or more complicated types**.

Variables

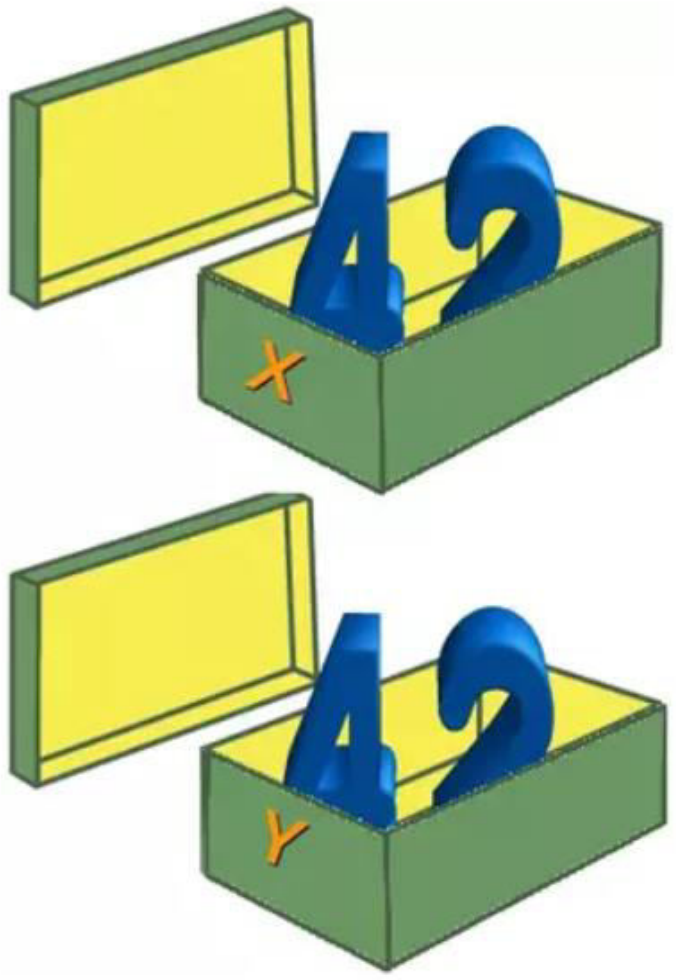
- A variable can be seen as a **container** (or some say a pigeonhole) to **store certain values**.
- While the program is running, variables are **accessed and sometimes changed**, i.e., a new value will be assigned to a variable.



Variables

- **Putting values** into the variables can be realized with **assignments**.
- In most cases, the **equal "=" sign** is used.
- The **value on the right side** will be saved in the **variable name on the left side.**
- Eg: x=42

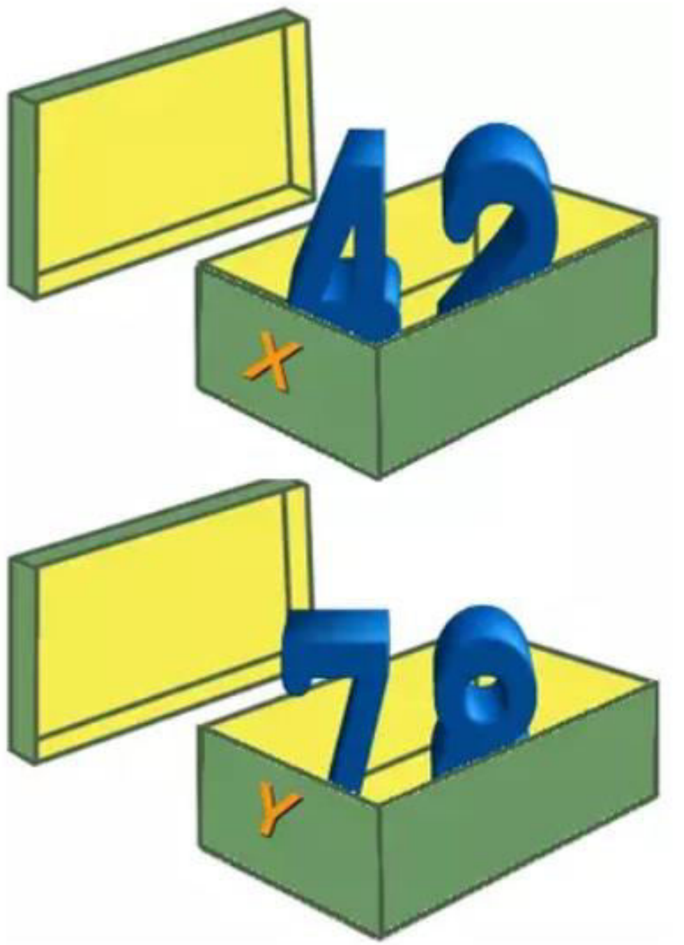
  y=42



Variables

- If we **assign a new value** to one of the variables, let's say the value 78 to y:

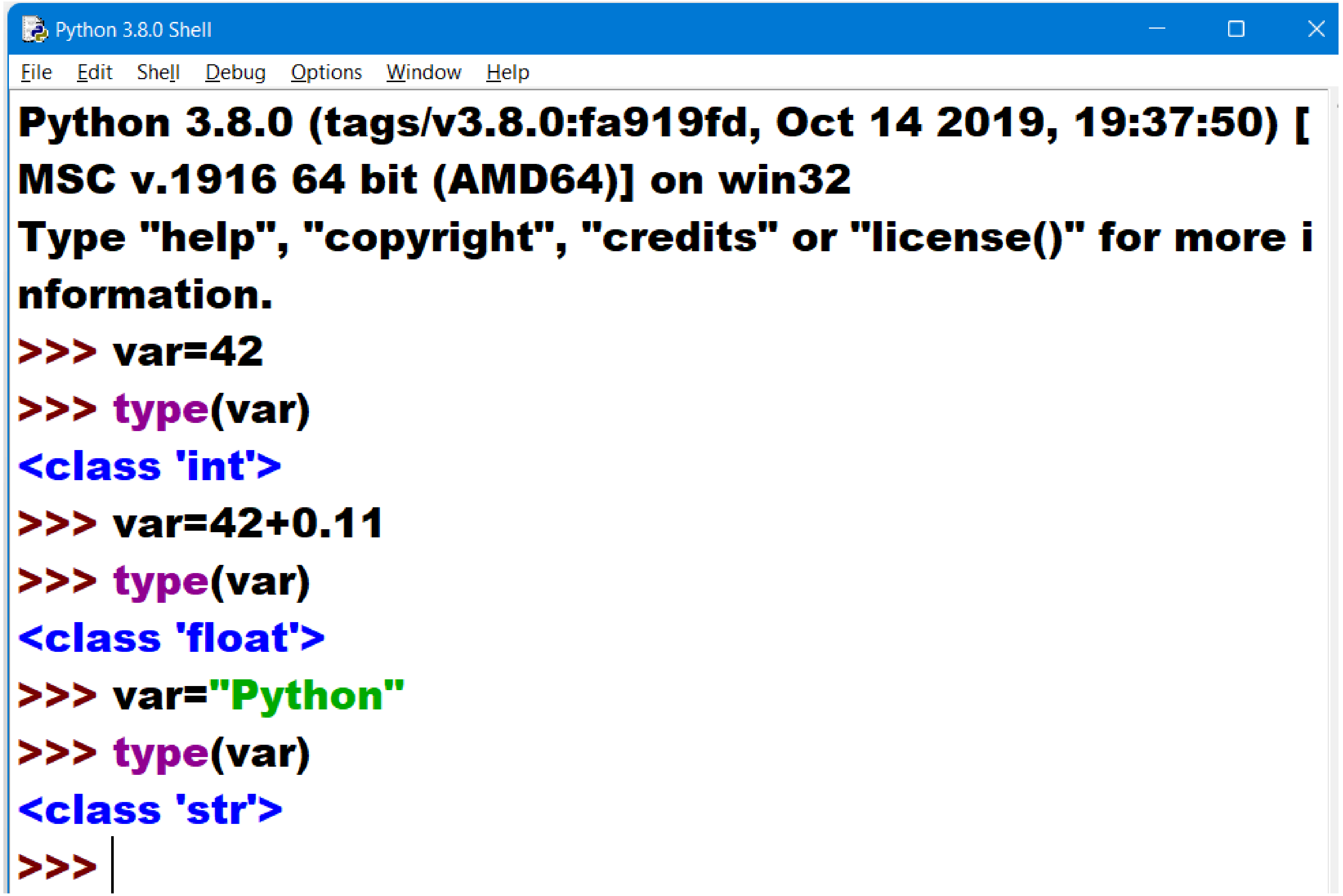
```
y = 78;
```



Variables

- There is **no declaration of variables** required in Python, which makes it quite easy.
- Not only the value of a variable may **change** during program execution, but the **type** as well.
- You can **assign an integer value** to a variable, use it as an integer for a while and then assign a **string to the same variable**.

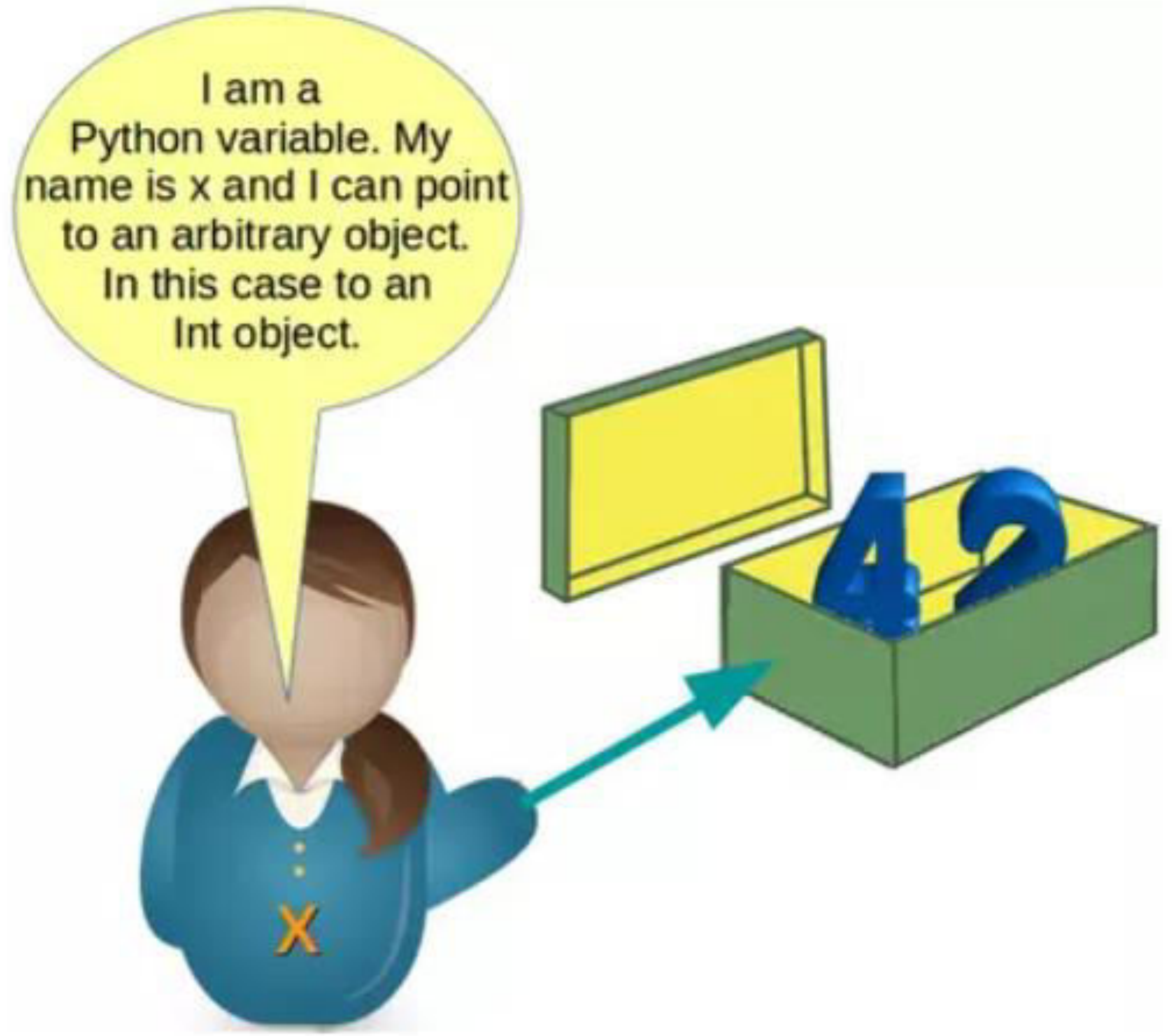
Variables



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [
MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more i
nformation.
>>> var=42
>>> type(var)
<class 'int'>
>>> var=42+0.11
>>> type(var)
<class 'float'>
>>> var="Python"
>>> type(var)
<class 'str'>
>>>
```

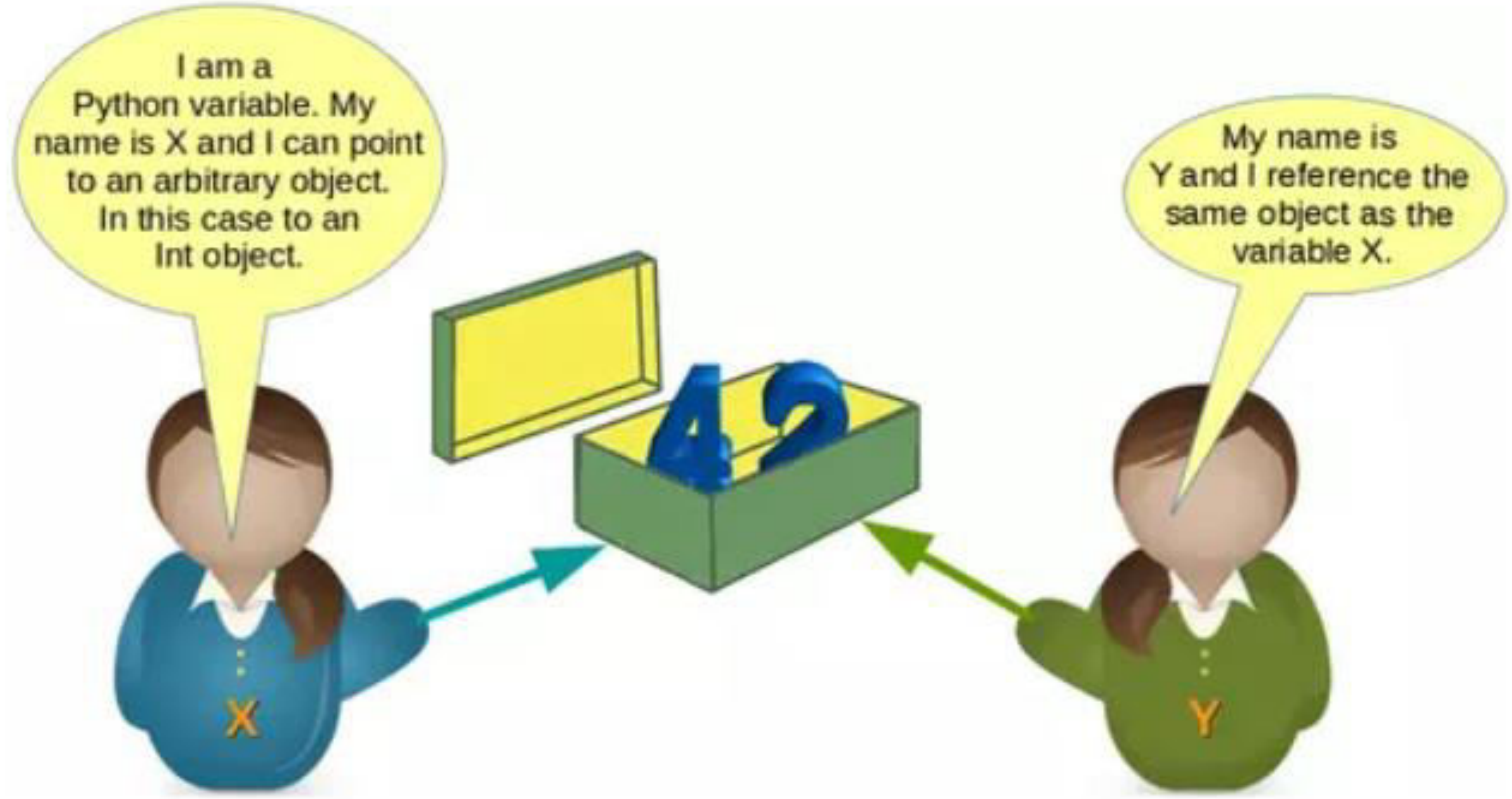
Object References

- Python **variables are references to objects**, but the actual data is contained in the objects:


I am a Python variable. My name is x and I can point to an arbitrary object. In this case to an Int object.
42
X

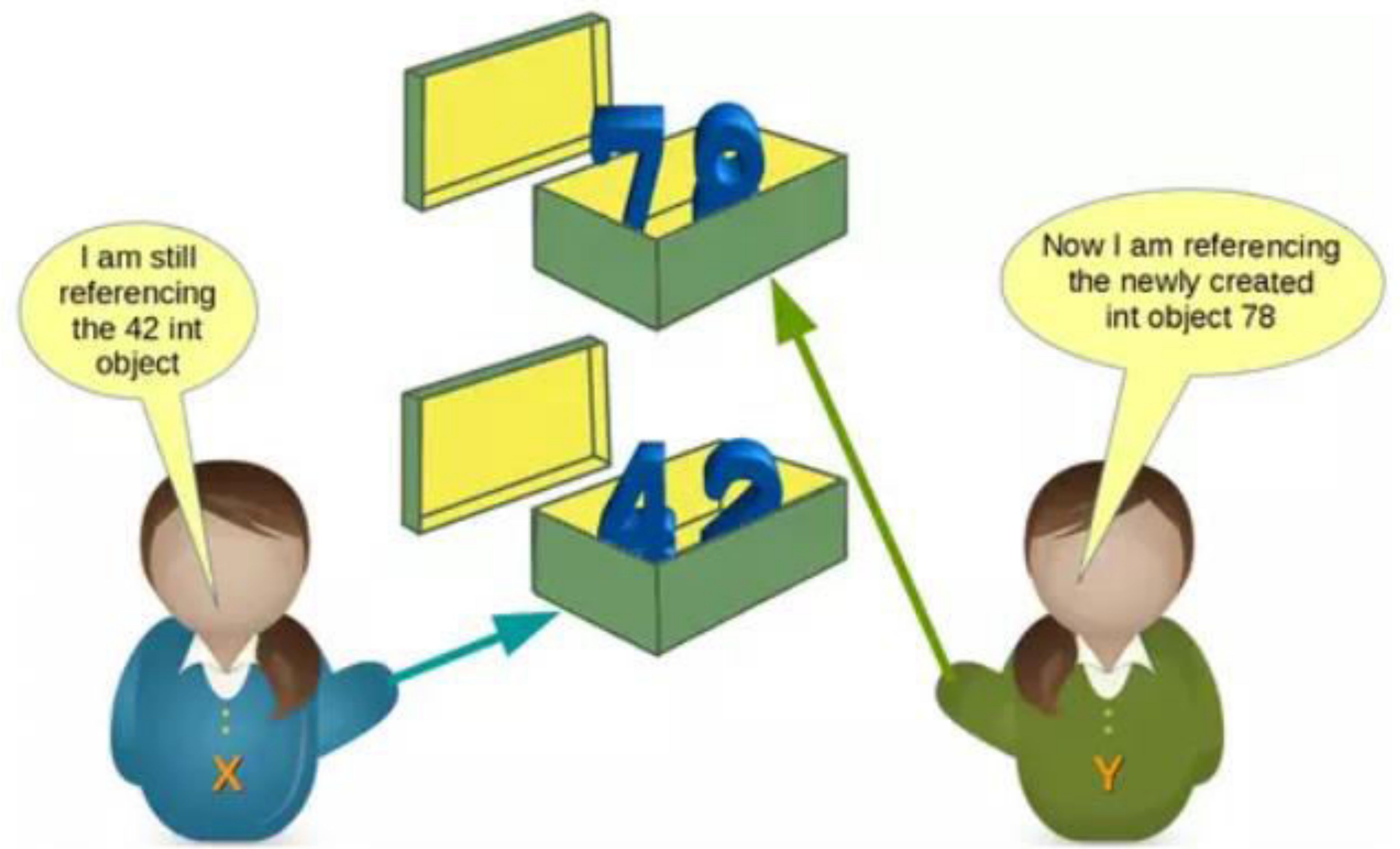
Object References

- x=42
- y=x


I am a Python variable. My name is X and I can point to an arbitrary object. In this case to an Int object.
My name is Y and I reference the same object as the variable X.
42
X
Y

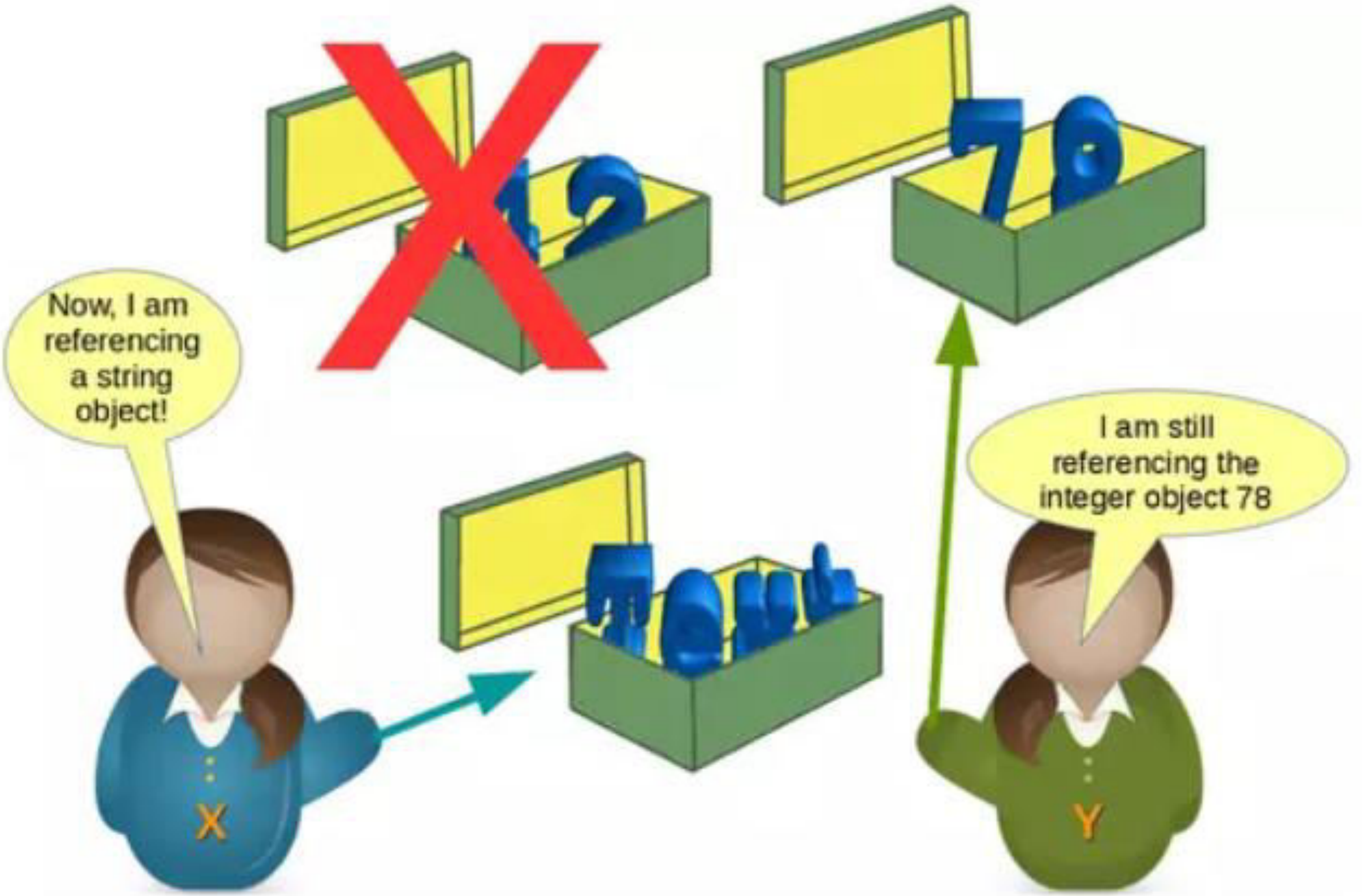
Object References

- y=78


I am still referencing the 42 int object
78
42
Now I am referencing the newly created int object 78
X
Y

Object References

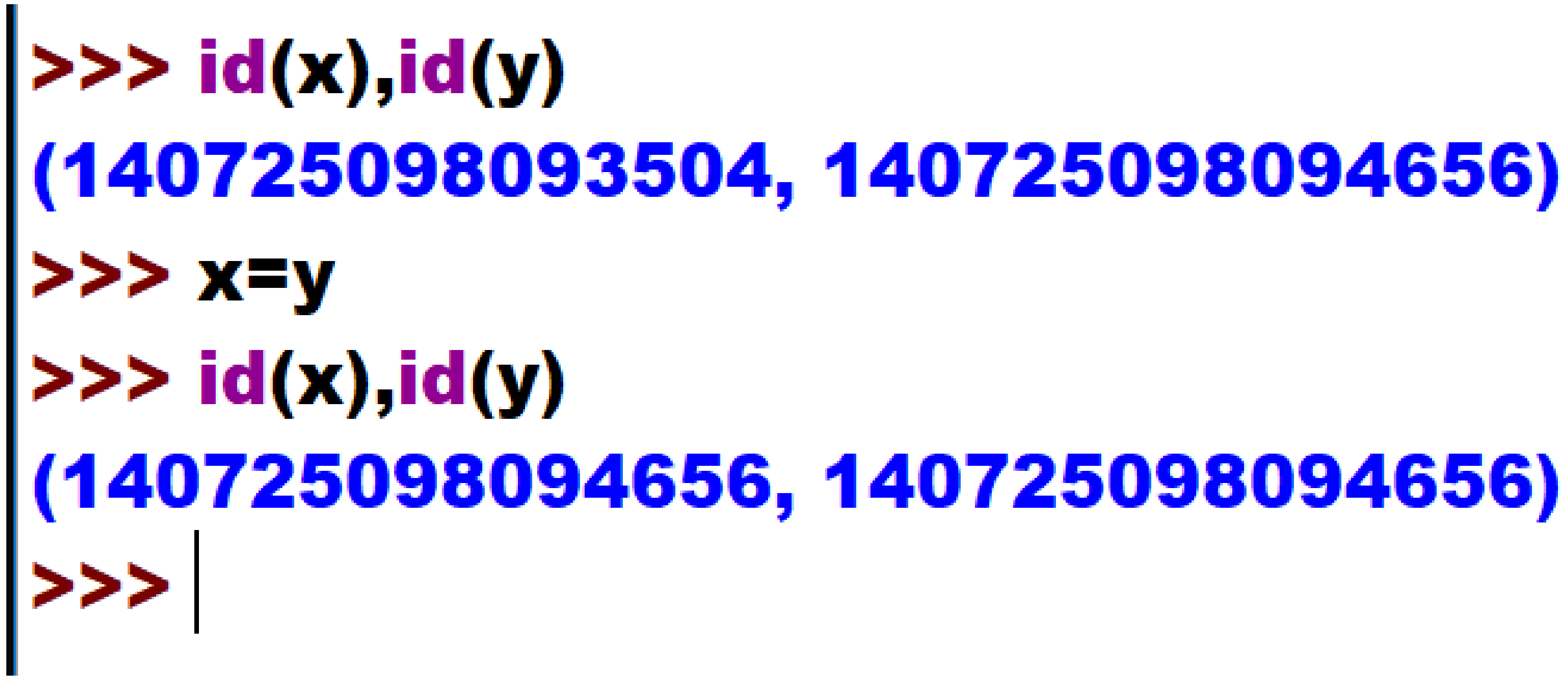
- x refers to some string values.


Now, I am referencing a string object!
I am still referencing the integer object 78
12
78
X
Y

Variables

- How can we see or prove that x and y really reference the same object after the assignment y = x of our previous example?
- Every **instance (object or variable) has an identity**, i.e., an integer which is unique within the script or program, i.e., other **objects have different identities**.
- The **identity function id()** can be used for this purpose.

Variables



```
>>> id(x),id(y)
(140725098093504, 140725098094656)
>>> x=y
>>> id(x),id(y)
(140725098094656, 140725098094656)
>>> 
```

Valid Variable Names

- The naming of variables follows the more general concept of an identifier.
- A Python **identifier** is a name used to **identify a variable, function, class, module or other object**.
- There are some certain rules to keep in mind that we must follow while naming identifiers.

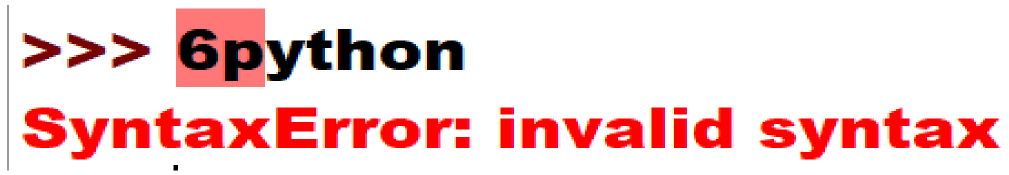
Rules for Naming Identifier

1. The Python identifier is made with a **combination of lowercase or uppercase letters, digits or an underscore**.

- These are the valid characters.
  - **Lowercase letters (a to z)**
  - **Uppercase letters (A to Z)**
  - **Digits (0 to 9)**
  - **Underscore (_)**
- Example: num1, FLAG, get_user_name, userDetails, _1234

Rules for Naming Identifier

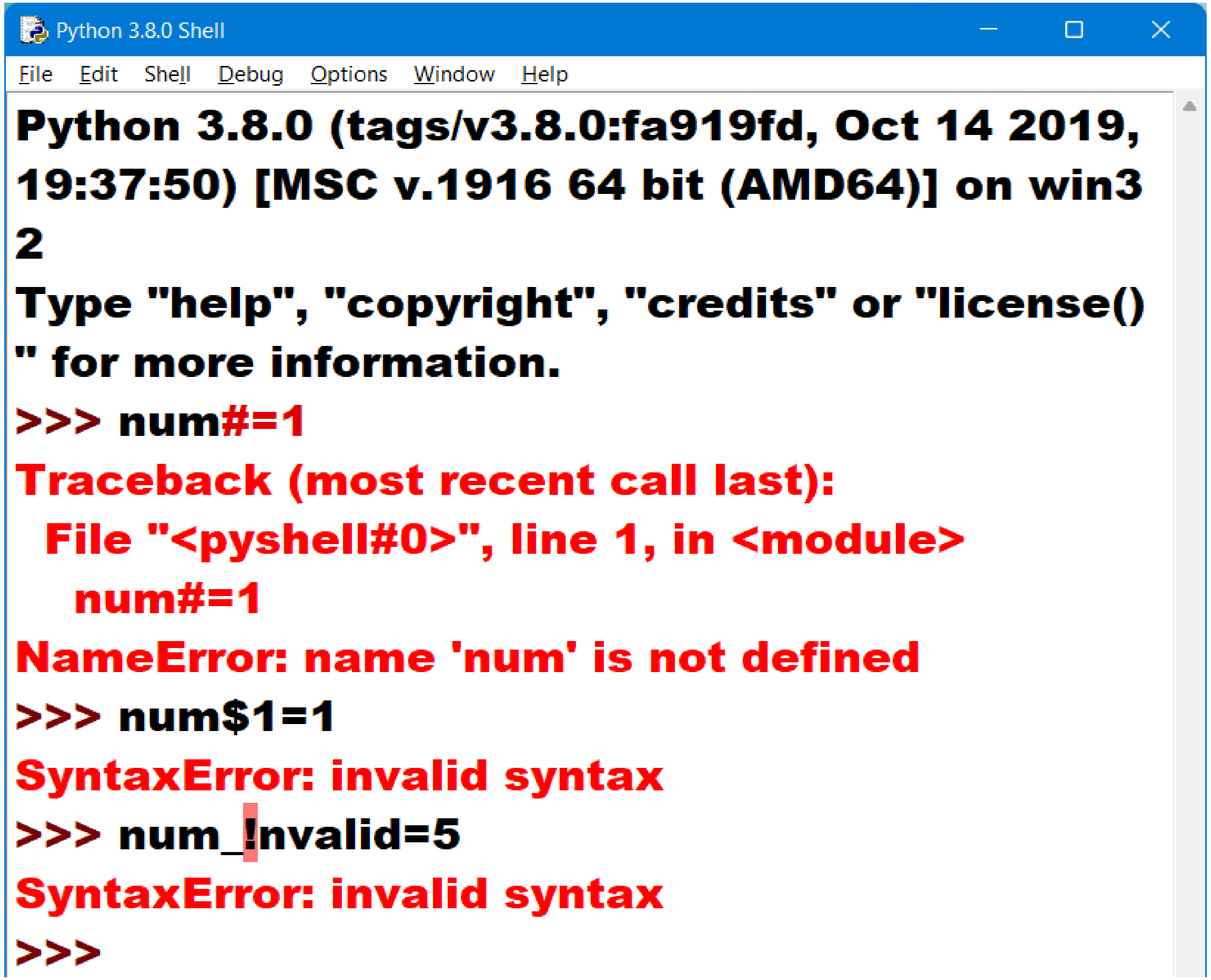
2. An identifier **cannot start with a digit**. If we create an identifier that starts with a digit then we will get a syntax error.



```
>>> 6python
SyntaxError: invalid syntax
```

3. We also **cannot use special symbols** in the identifiers name. Symbols like **(!, @, #, $, %, .) are invalid**.

Rules for Naming Identifier

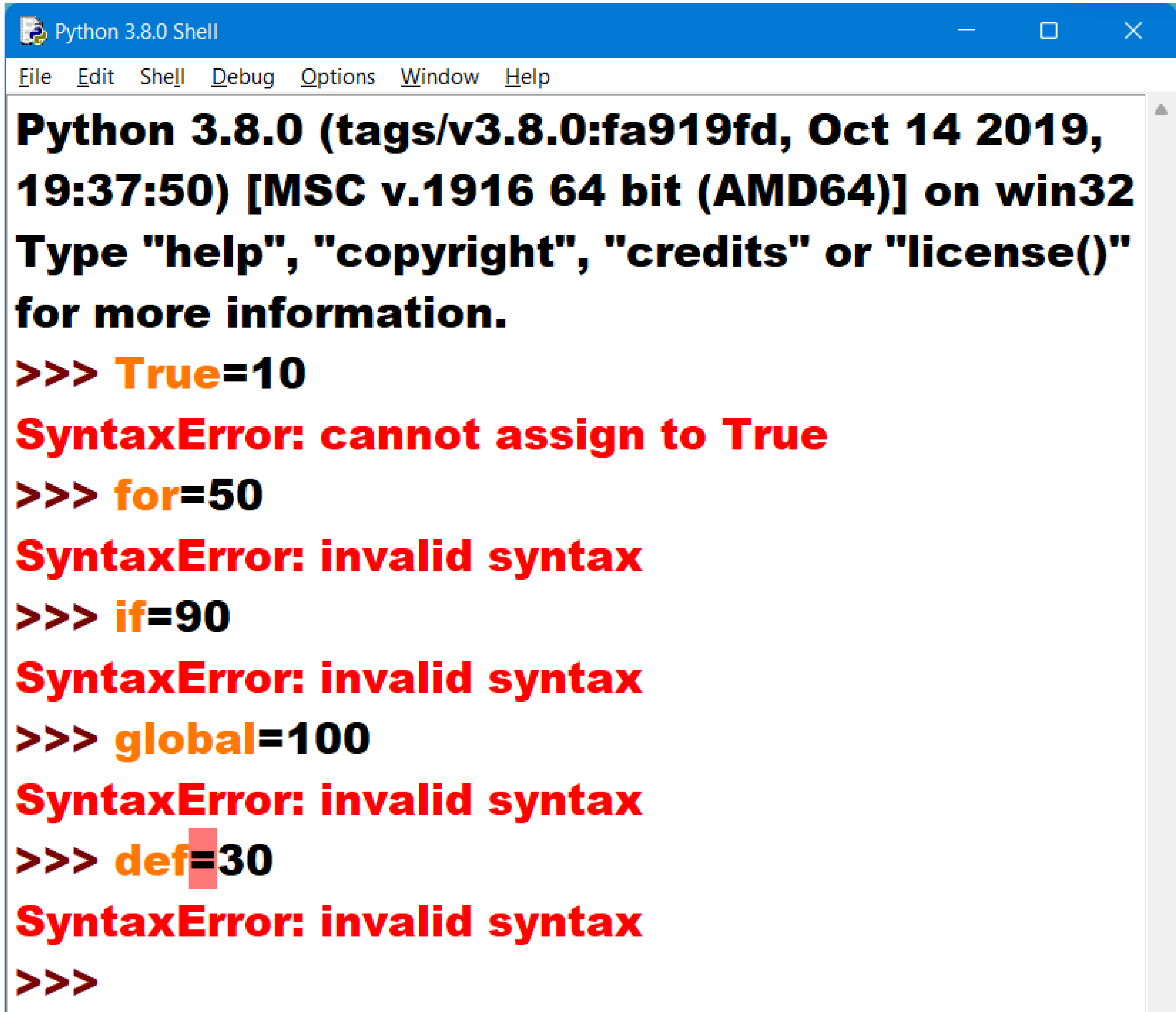


```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019,
19:37:50) [MSC v.1916 64 bit (AMD64)] on win3
2
Type "help", "copyright", "credits" or "license()
" for more information.
>>> num#=1
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    num#=1
NameError: name 'num' is not defined
>>> num$1=1
SyntaxError: invalid syntax
>>> num_!nvalid=5
SyntaxError: invalid syntax
>>>
```

Rules for Naming Identifier

4. A **keyword cannot** be used as an identifier.

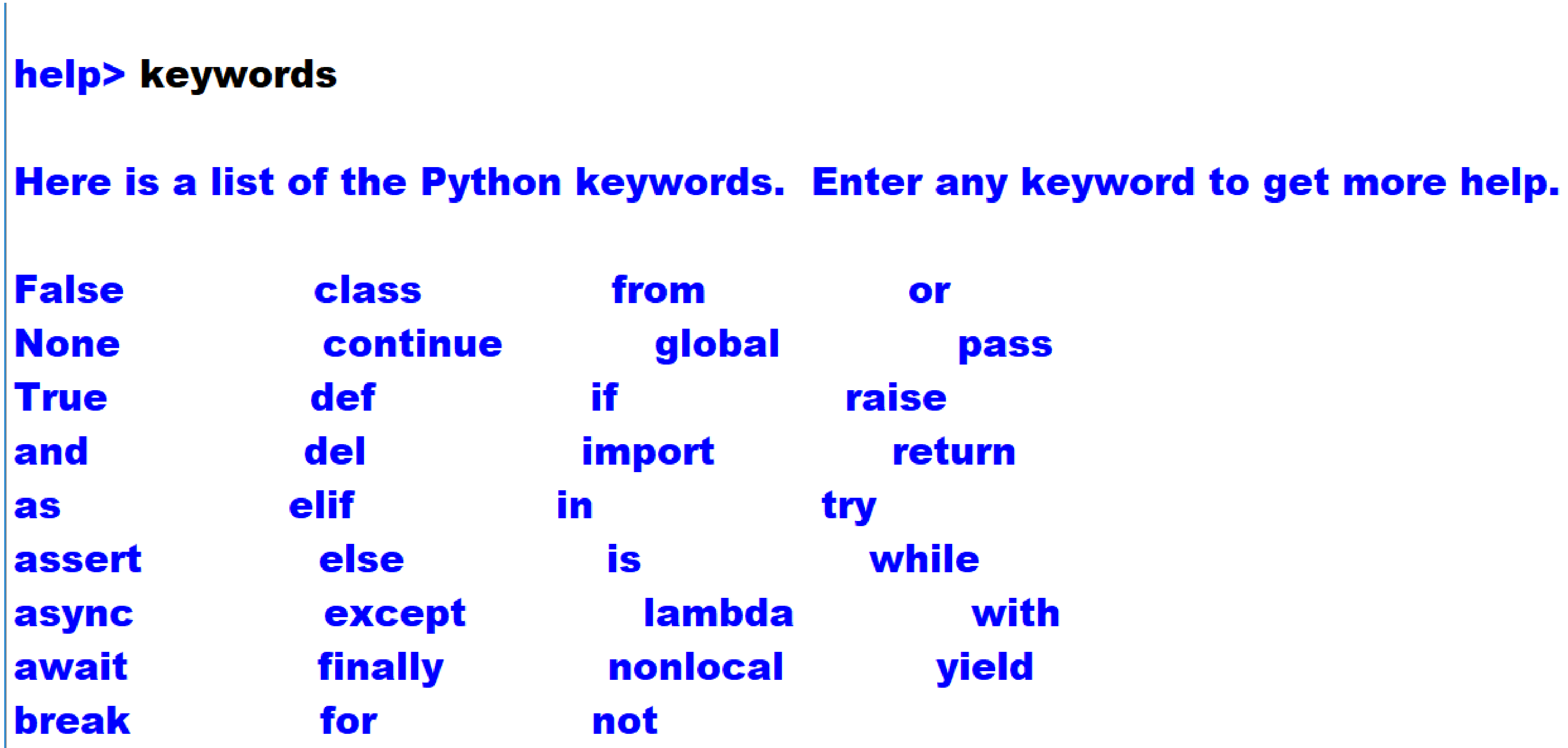
In Python, **keywords** are the **reserved names** that are built-in in Python. They have a **special meaning**, and we cannot use them as identifier names.



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019,
19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()"
for more information.
>>> True=10
SyntaxError: cannot assign to True
>>> for=50
SyntaxError: invalid syntax
>>> if=90
SyntaxError: invalid syntax
>>> global=100
SyntaxError: invalid syntax
>>> def=30
SyntaxError: invalid syntax
>>>
```

Rules for Naming Identifier

- If you want to see the list of all the keywords, then in your Python shell, type **“help()”** and then type **“keywords”** to get the list of all Python keywords.



```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

Rules for Naming Identifier

- The **length** of the identifiers can be **as long as you want**.
- Of course, it can **not be greater** than the **available memory**, however, the **PEP-8 standards rule** suggests not to exceed **79 characters** in a line.

Best Practices for Python Identifiers

1. **Class names** should start with a **capital letter** and all the **other identifiers** should start with a **lowercase letter**.

2. Begin **private identifiers** with an **underscore (_)**. Note, this is not needed to make the variable private. It is only for the ease of the programmer to easily **distinguish** between **private variables and public variables**.

3. Use **double underscores (__)** around the **names of magic methods** and don't use them anywhere else. Python **built-in magic methods** already use this notation. For example: __init__ , __len__ .

Best Practices for Python Identifiers

4. Always prefer using **names longer than one character**. index=1 is better than i=1

5. To **combine words** in an identifier, you should **use underscore(_)**. For example: get_user_details.

6. Use **camel case** for naming the variables. For example: fullName, getAddress, testModeOn, etc.