變數

- In statically typed languages, such as C, a variable must have a predefined type, and this variable can only hold the value of this predefined type.
- In Python, a variable can be reused to store values of other types.
- In Python, a new variable is defined by assigning a value to the variable.

In [1]:

```
sum = 0 # define one variable
x, sum, product = 0, 0, 1 # define three variables
```

Scope and Lifetime ¶

- The part of a program where a variable is accessible is the scope of this variable.
- The duration for which a variable exists is the lifetime of this variable.
- A variable which is defined in the main body of a file is called a global variable. This variable is visible throughout the file and also inside any file which imports that file.
- A variable defined inside a function is local to this function. This variable is accessible from the point at which it is defined until the end of the function.

In [1]:

```
# This is a global variable
a = 0

if a == 0:
    # This is still a global variable
    b = 1

def my_function(a):
# they are local variables
    d = 3
    print('local a',a)
    print('local d',d)

# Now we call the function and pass the value 7 as the first and only parameter
my_function(7)
# a and b still exist
print('global a',a)
print('global b',b)

# c and d don't exist anymore -- these statements will give us name errors!
print('undefined c',c)
print('local d',d)
```

```
local a 7
local d 3
global a 0
global b 1

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-d96de9e18af6> in <module>()
     19
     20 # c and d don't exist anymore -- these statements will give us name
 errors!
---> 21 print('undefined c',c)
     22 print('local d',d)

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

Scope

- Global variables

In [3]:

```
#global variable
a = 0
def my_function1():
    print(a)

my_function1()
print(a)
```

```
0
0
```

In [4]:

```
a = 0
def my_function2():
    global a
    a=3
    print(a)
my_function2()
print(a)
```

```
3
3
```

In [3]:

```
a = 0
def my_function3():
    print(a)
    global a
    a=3
    print(a)
my_function3()
print(a)
```

```
0
3
3

<ipython-input-3-cf4b58e6e910>:4: SyntaxWarning: name 'a' is used prior to g
lobal declaration
  global a
```

In [2]:

```
def my_function4():
    global aa
    print(aa) # refer to an undefined global variable

my_function4()
print(aa)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-a24197731575> in <module>()
      3     print(aa) # refer to an undefined global variable
      4
----> 5 my_function4()
      6 print(aa)

<ipython-input-2-a24197731575> in my_function4()
      1 def my_function4():
      2     global aa
----> 3     print(aa) # refer to an undefined global variable
      4
      5 my_function4()

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

- Local variables

In [7]:

```
a = 0
def my_function6():
    a=3 # this is a local variable
    print(a)

my_function6()
print(a)
```

```
3
0
```

In [8]:

```
a = 0
def my_function7():
    print(a) # a is a local variable but is referred before it is created.
    a=3
    print(a)
my_function7()
print(a)
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-8-bd1c817ee6df> in <module>()
      4     a=3
      5     print(a)
----> 6 my_function7()
      7 print(a)

<ipython-input-8-bd1c817ee6df> in my_function7()
      1 a = 0
      2 def my_function7():
----> 3     print(a) # a is a local variable but is referred before it is cr
eated.
      4     a=3
      5     print(a)

UnboundLocalError: local variable 'a' referenced before assignment
```

In [9]:

```
a = 0
def my_function7():
    a=3
    def my_inner_function():
        nonlocal a
        a = 6
        print('my_inner_function:',a)
    print('my_function7-1:',a)
    my_inner_function()
    print('my_function7-2:',a)

my_function7()
print('global:',a)
```

```
my_function7-1: 3
my_inner_function: 6
my_function7-2: 6
global: 0
```

Constants

- Python does not have constant variables. In Python, the name of a variable whose value is not meant to change is usually in all caps with underscores separating words.

```
MAXIMUM_NUMBER = 20
#We define some options which are constants
LOWER, UPPER, CAPITAL = 1, 2, 3
name = "jane"
#We use our constants when assigning these values...
print_style = UPPER
#...and when checking them:
if print_style == LOWER:
    print(name.lower())
elif print_style == UPPER:
    print(name.upper())
elif print_style == CAPITAL:
    print(name.capitalize())
else:
    print("Unknown style option!")
```

- Some Python modules define common constants.

```
import string, math, re

#all the lowercase ASCII letters
print(string.asci_lowercase) # 'abcdefghijklmnopqrstuvwxyz'

#some mathematical constants
print(math.e) # 2.718281828459045
print(math.pi) # 3.141592653589793
```

Mutable Types and Immutable Types

- Statically typed or Dynamically typed
- Variable scope and lifetime
- Type conversion
- We can only assign a variable of an immutable type a new value. Integers, floating-point numbers, and strings are all immutable types.
- We can alter the value of a variable of a mutable type. Lists, dictionaries, and most of objects are mutable.
  - Examples

```
my_list = [1,2,3,4]
my_list[0] = 0 # we can change the first element of this list
class myClass(object): # declare a class
    pass;
my_object = myClass();
my_object.a_property = 99; # We can change the value of the attribute of my
_object
```

Type Conversion

- Convert data from one type to another type
- Implicit conversion

- Integer numbers can be converted to floating-point numbers implicitly.

```
result = 2 + 3//2 + 2.5  # the result is a floating-point number
if result:
    print("result is not equal to 0")
if 3 > 4: # convert to a Boolean value
    print("This line cannot be reached")
print(3) # implicitly convert to "3"
```

- Explicit conversion
  - Examples

```
int("5") # correct
int(5.5) # correct
int("5.5") # incorrect
int(float("5.5")) # correct

str(3)   # convert to "3"
str(3.3)  # convert to "3.3“
int("3")+20 # convert to 23
"3"+str(10) # convert to "310"
"hello" + 3  # incorrect
"hello%d"%3  # correct, and get "hello3"
```

In [10]:

```
from math import ceil, floor
x = ceil(2.4) # x is equal to 3
x = floor(2.9) # x is equal to 2
x = round(2.6) # x is equal to 3
```

In [11]:

```
try:
    age = int(input("Enter your age:"))
except ValueError as e:
    print("Error reading age: {}".format(e))
```

```
Enter your age:21a
Error reading age: invalid literal for int() with base 10: '21a'
```