Python变量

Python变量不用指定类型，解释器会自动推断变量的数据类型

02-Python变量、简单数据类型和列表

In [1]:

```
message = 'Hello World!'
numbers = 100

print(message)
print(numbers)
```

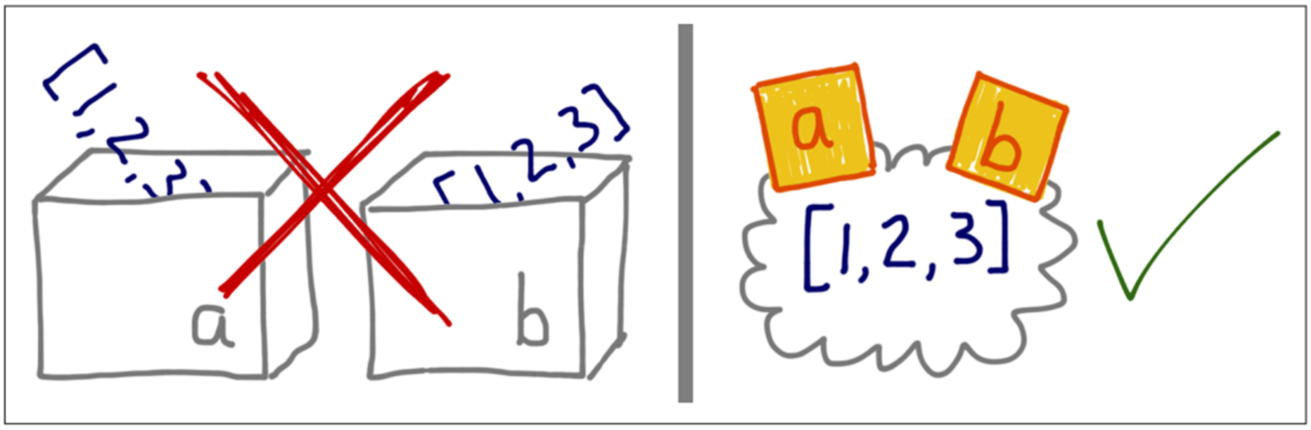
```
Hello World!
100
```

变量随时可以再被赋予任意类型的值:

In [2]:

```
message = 42
numbers = 'Hello World!'
```

变量不是盒子，而是标签



变量命名规则:

- 使用英语单词： my_message, first_name
- 变量名称由数字、字母(包括大写字母和小写字母)、下划线组成。
- 变量名不能以数字开头
- 变量名不能用python关键字（p426 附录A.4）
- 变量命名严格区分大小写

字符串

字符串就是一系列字符，使用单引号或者双引号扩起来

In [3]:

```
message = 'hello'
name = "ada lovelace"
```

Python语言中没有区分字符（char）和字符串（str），字符就是长度为1的字符串

字符串的方法：

- title方法：将单词的首字母大写
- 将字符串改为全部大写或者小写: upper(), lower()
- format方法或者f字符串：字符串的格式化
- 字符串中的转义字符：制表符'\t', 换行'\n'
- 删除空白: rstrip(), lstrip(), strip()

将单词的首字母大写

In [4]:

```
name.title()
```

Out[4]:

```
'Ada Lovelace'
```

将字符串改为全部大写或者小写: upper(), lower()

In [5]:

```
print(name.upper())
print(name.lower())
```

```
ADA LOVELACE
ada lovelace
```

f 字符串

In [6]:

```
first_name = 'ada'
last_name = 'lovelace'
full_name = f'{first_name} {last_name}' # Python 3.6+
print(full_name)
```

```
ada lovelace
```

In [7]:

```
full_name = '{} {}'.format(first_name, last_name)
print(full_name)
```

```
ada lovelace
```

使用制表符或换行符添加空白

In [8]:

```
print('Languages:\n\tPython\n\tC\n\tJavaScript')
```

```
Languages:
	Python
	C
	JavaScript
```

删除空白

In [9]:

```
favorite_language = 'python '
print(favorite_language)
```

```
python
```

In [10]:

```
favorite_language.rstrip()
```

Out[10]:

```
'python'
```

In [11]:

```
favorite_language = ' python '
favorite_language.rstrip()
```

Out[11]:

```
' python'
```

In [12]:

```
favorite_language.lstrip()
```

Out[12]:

```
'python '
```

```
In  [13]:
```

```
favorite_language.strip()
```

```
Out[13]:
```

```
'python'
```

避免字符串的语法错误

```
In  [14]:
```

```
message = 'One of Python's strengths is its diverse community.'
```

```
  File "C:\Users\zhouj\AppData\Local\Temp/ipykernel_15056/2221409680.
py", line 1
    message = 'One of Python's strengths is its diverse community.'
                             ^
SyntaxError: invalid syntax
```

```
In  [15]:
```

```
message = "One of Python's strengths is its diverse community."
print(message)
```

```
One of Python's strengths is its diverse community.
```

```
In  [16]:
```

```
message = "One of Python\'s strengths is its diverse community."
print(message)
```

```
One of Python's strengths is its diverse community.
```

数

- 整数（int）： 没有区分长度（没有int32， int64，long），从Python 3.8开始没有最大值的限制
- 浮点数(float): 没有区分单精度和双精度
- int和float的实际长度会根据机器平台来决定，绝大多数情况下为64位，8个字节

这里讲解的所有的运算都可以使用整数和浮点数

基本运算： +, -, *, /

In [17]:

```
2 + 3
```

Out[17]:

```
5
```

In [18]:

```
3 - 2
```

Out[18]:

```
1
```

In [19]:

```
3 * 2.5
```

Out[19]:

```
7.5
```

In [20]:

```
3 / 1 # 结果一定是浮点数
```

Out[20]:

```
3.0
```

乘方运算 **

In [21]:

```
3 ** 2
```

Out[21]:

```
9
```

In [22]:

```
3 ** 0.5
```

Out[22]:

```
1.7320508075688772
```

In [23]:

```
0 ** 0
```

Out[23]:

```
1
```

模运算： % （得到余数）

In [24]:

```
5 % 3
```

Out[24]:

```
2
```

In [12]:

```
5.25 % 1 # 浮点数的小数部分
```

Out[12]:

```
0.25
```

除法求商：//

In [26]:

```
10 // 3
```

Out[26]:

```
3
```

In [27]:

```
5.25 // 1 # 浮点数的整数部分
```

Out[27]:

```
5.0
```

求商和余数： divmod 函数

In [13]:

```
divmod(10, 3)
```

Out[13]:

```
(2.0, 3.0)
```

round 函数: 浮点数四舍五入

In [29]:

```
round(0.666)
```

Out[29]:

```
1
```

In [30]:

```
round(0.333)
```

Out[30]:

```
0
```

In [31]:

```
round(0.5)
```

Out[31]:

```
0
```

习题：求离整数n最近的平方数

例如，如果n=111，那么nearest_sq(n)等于121，因为111比100（10的平方）更接近121（11的平方）。

如果n已经是完全平方（例如n=144，n=81，等等），你需要直接返回n。

In [32]:

```
def nearest_sq(n):
    return round(n ** 0.5) ** 2

nearest_sq(111)
```

Out[32]:

```
121
```

任务: 给出一个整数，确定它是否是一个平方数。

```
-1 => False
 0 => True
 3 => False
 4 => True
25 => True
26 => False
```

In [15]:

```
def is_square(n):
    return n>=0 and n**0.5 % 1 == 0

is_square(-1)
is_square(144)
```

Out[15]:

```
True
```

为什么0.3+0.1不等于0.4?

In [34]:

```
# In Python 3.10, 0.3+0.1==0.4 is True
.3 + .1 == .4
```

Out[34]:

```
True
```

In [35]:

```
.3 + .1 + .2 == .6
```

Out[35]:

```
False
```

数字中的下划线

In [36]:

```
universe_age = 14_000_000_000
```

同时给多个变量赋值

In [37]:

```
x, y, z = 0, 0, 0
```

常量：常量名应该全部大写

In [38]:

```
MAX_CONNECTIONS = 5000
```

Python语法没有强制约定常量不能被修改

In [39]:

```
MAX_CONNECTIONS = 15000
```

代码注释

In [40]:

```
# 向大家问好。
print("Hello Python people!")
```

```
Hello Python people!
```

In [41]:

```
print("Hello Python people!") # 向大家问好。
```

```
Hello Python people!
```

字符串和数字之间的转化

- str()函数：将其他数据转化为字符串
- int()函数：将其他数据转化为整数
- float()函数：将其他数据转化为浮点数

In [5]:

```
str1 = str(123)
str1
```

Out[5]:

```
'123.45'
```

In [7]:

```
int1 = int('123')
int1
```

Out[7]:

```
3
```

In [8]:

```
int2 = int('123.4')
int2
```

```
---------------------------------------------------------------------------
------
ValueError                                Traceback (most recent call last)
Cell In[8], line 1
----> 1 int2 = int('123.4')
      2 int2

ValueError: invalid literal for int() with base 10: '123.4'
```

In [9]:

```
f1 = float('123.4')
f1
```

Out[9]:

```
123.4
```

In [10]:

```
f2 = float('123')
f2
```

Out[10]:

```
123.0
```

三个关于变量的函数

- type函数: 返回该变量的类型
- id函数： 返回该函数的id，这是一个int类型的值
- isinstance函数：如果该变量是某类型的实例，返回True，否则返回False

In [42]:

```
message = "Hello"
number = 42
pi = 3.14159
```

In [43]:

```
print(id(message), id(number), id(pi), sep=", ")
```

```
1902389825584, 1902310026832, 1902389782128
```

In [44]:

```
print(type(message), type(number), type(pi), sep=', ')
```

```
<class 'str'>, <class 'int'>, <class 'float'>
```

In [45]:

```
print(isinstance(message, str), isinstance(number, int), isinstance(pi, float),sep=', ')
```

```
True, True, True
```

简单类型变量都是不可变的

In [46]:

```
number = 42
print(id(number))
number = 100
print(id(number))
```

```
1902310026832
1902310217168
```

Python之禅

In [47]:

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Type *Markdown* and LaTeX: α^2

In [ ]:

```

```