2. Variables

《Python programming》 / Lecturer：Zhiyi Luo (罗志一)

School of Computer Science and Technology
计算机科学与技术学院

浙江理工大学
ZHEJIANG SCI-TECH UNIVERSITY

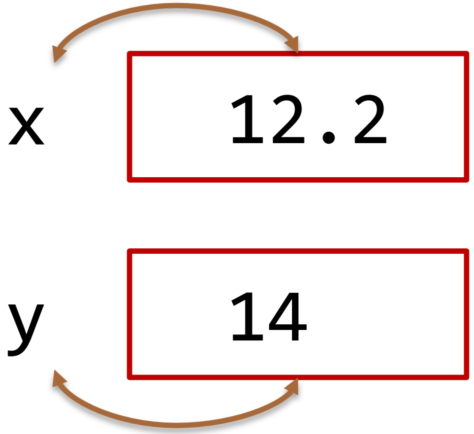
Overview

- Variables
- Expressions
- Statements

Variables (Name)

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

```
x = 12.2
y = 14
```

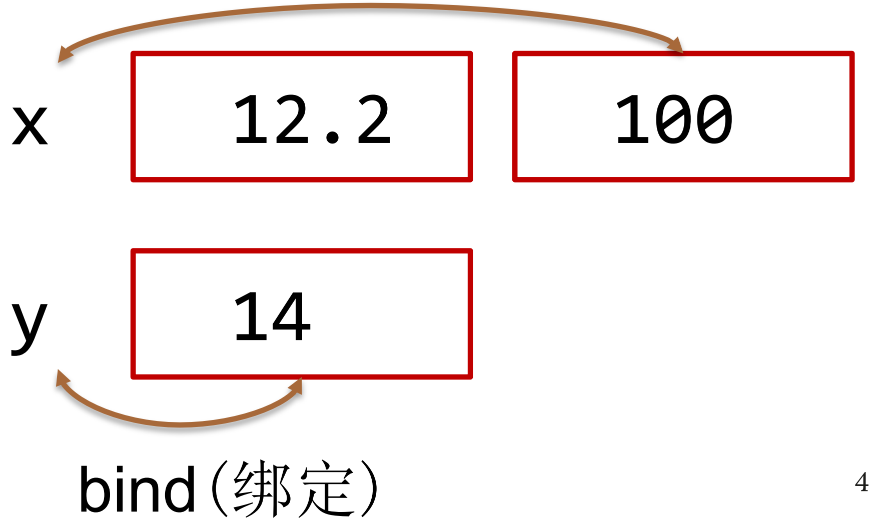


bind (绑定)

Variables (Name)

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable "name"
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

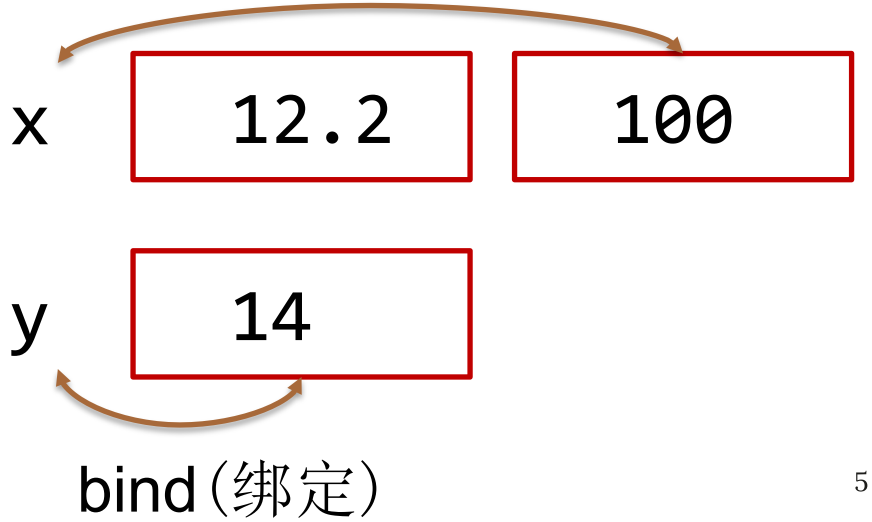
```
x = 12.2
y = 14
x = 100
```



Name binding（名称绑定）

- Name binding is the association between a name and an object (value). So in Python, we bind (or attach) a name to an object.
- One way to bind a name to an object is to use the assignment operator (=).

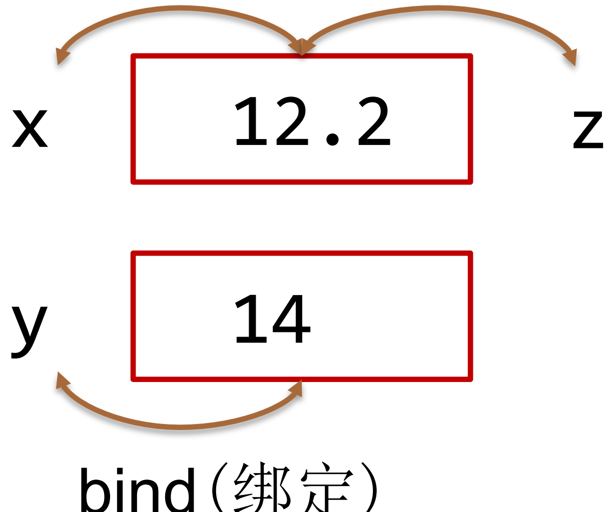
```
x = 12.2
y = 14
x = 100
```



Name binding（名称绑定）

- When you enter a name in Python, it gives you back the object bound to it.
- Have a try !

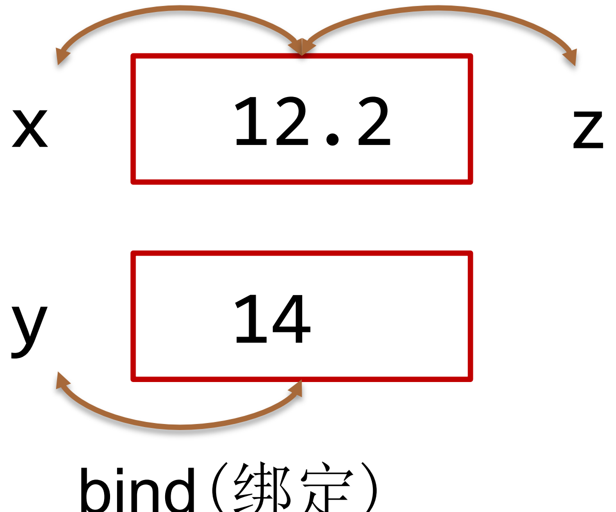
```
x = 12.2
y = 14
z = x
```



Name binding（名称绑定）

- What happens when we use a name on the right-hand side of the = operator ?
- For example, z = x means bind the name z to the object bound to the name x.

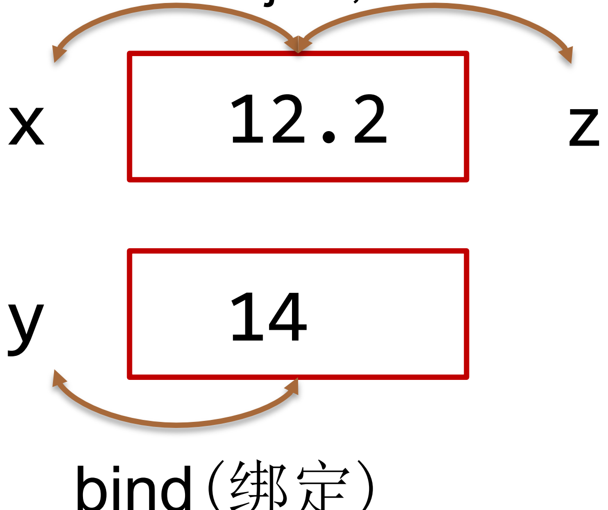
```
x = 12.2
y = 14
z = x
```



Name binding（名称绑定）

- Python create a name z and bound it to the object 12.2, the object bound to x. Note that Python didn't create or copy the object 12.2, it only introduced a new name for that object. So both x and z are bound to the object 12.2.
- Remember, a name can only be bound to an object, not to a name.

```
x = 12.2
y = 14
z = x
```



Data Types

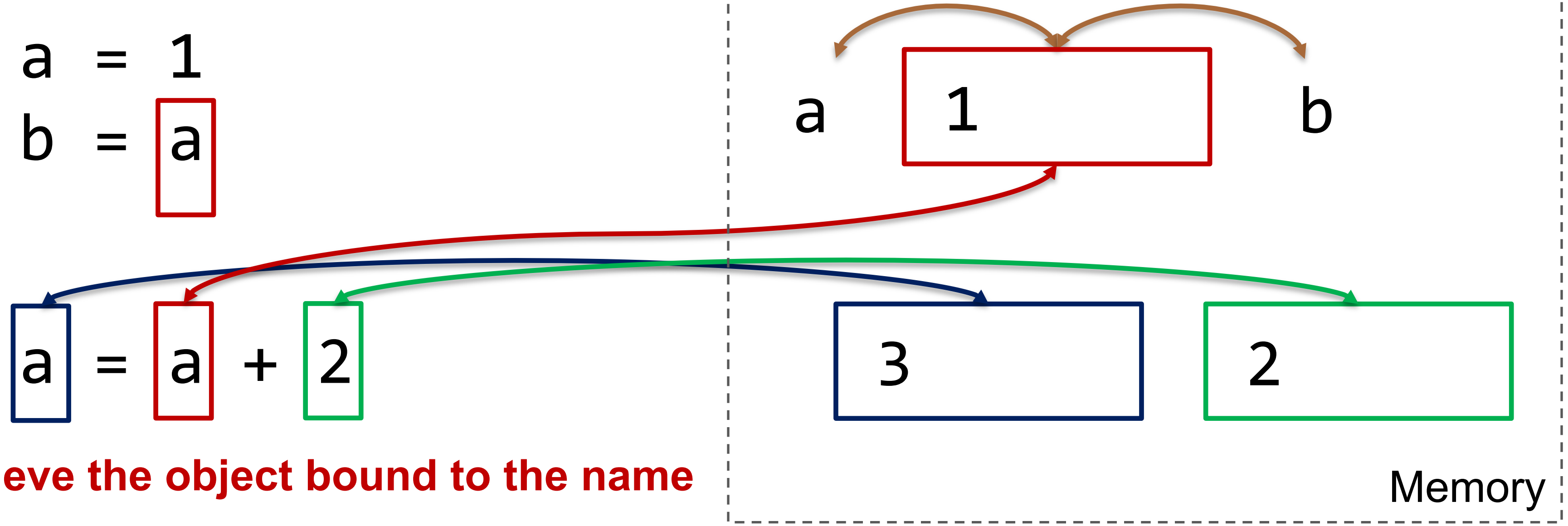
	Type	Mutable	Examples
Numbers	bool	No	True, False
	int	No	13, 256,1024
	float	No	1.21, 3.14, 2e-7
	complex	No	5+9j
String	str	No	“hello”, ‘Jack’
Bytes	bytes	No	b‘ab\xff’
List	list	Yes	[‘Winken’, ’Blinken’, ’Nod’]
Tuple	tuple	No	(2, 4, 8)
Dictionary	dict	Yes	{“name”: “Jack”, “age”: 18}
Set	set	Yes	Set([3, 5, 7])

Mutable & Immutable

- Mutable: can be changed
- Immutable: constant

Modifying an object

- So far we used immutable objects (floats and integers). Let's see an example with a list.



```
a = 1
b = a

a = a + 2
```

retrieve the object bound to the name

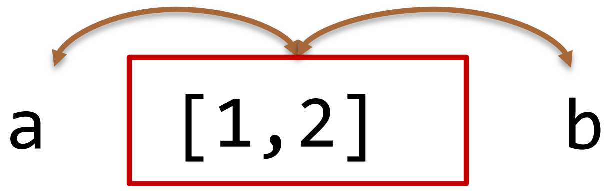
Modifying an object

```
Python 3.8.5 (default, Sep  4 2020, 02:22:02)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 1
>>> b = a
>>> id(a)
4302481760
>>> id(b)
4302481760
>>> a = a + 2
>>> id(a)
4302481824
```

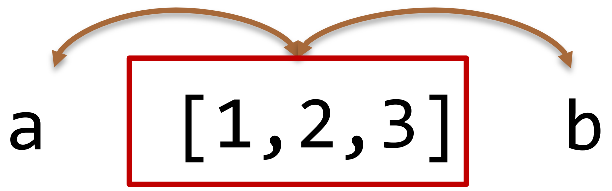
Modifying an object

- So far we used immutable objects (floats and integers). Let’s see an example with a list.

```
a = [1, 2]
b = a
```



```
a.append(3)
```



retrieve the object bound to the name

Modifying an object

```
Python 3.8.5 (default, Sep  4 2020, 02:22:02)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = [1, 2]
>>> b = a
>>> id(a)
140432277687744
>>> id(b)
140432277687744
>>> a.append(3)
>>> id(a)
140432277687744
>>> id(b)
140432277687744
```

Identifiers

- Identifiers: names given to variables
- The identifier is a combination of character digits and underscore and the character includes letters in lowercase (a-z), letters in uppercase (A-Z), digits (0-9), and an underscore (_).
- An identifier cannot begin with a digit. If an identifier starts with a digit, it will give a Syntax error.
- Special symbols like !, @, #, $, %, etc. are not allowed in identifiers.
- Python identifiers cannot only contain digits.
- There is no restriction on the length of identifiers.
- Identifier names are case-sensitive.

Keywords: Reserved Words

- You cannot use reserved words as variable names / identifiers

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

Identifiers Example

- **Python Valid Identifiers Example**
  - abc123, abc_de, _abc, ABC, abc

- **Python Invalid Identifiers Example**
  - 123abc, abc@, 123, for

Identifiers

- **Testing the Validity of Python Identifiers**
  - Use the **str.isidentifier()** function to check the validity of an identifier, but this method doesn't take reserved words into consideration. So, we can use this function with **keyword.iskeyword()** to check if the name is valid or not.

```
print ("abc".isidentifier())
print ("123abc".isidentifier())
print ("_abc".isidentifier())
print ("for".isidentifier())
```

Identifiers

- **Testing the Validity of Python Identifiers**
  - Use the **str.isidentifier()** function to check the validity of an identifier, but this method doesn't take reserved words into consideration. So, we can use this function with **keyword.iskeyword()** to check if the name is valid or not.

```
import keyword
def is_valid_identifier(x):
    Return x.isidentifier() and not keyword.iskeyword(x)
print(is_valid_identifier("for"))
```

Expressions

- An expression is the combination of variables and operators that evaluate based on operator precedence.
- Python expressions only contain identifiers, literals and operators.
- **literals**(字面常量)
  - Literals are notations for constant values of some built-in types.
  - String literal, bytes literal, integer, float number, image number
  - E.g. ‘hello’, b‘spam’, 100, 3.14, 1+3j

Mutable or Immutable?

Expressions

- An expression is the combination of variables and operators that evaluate based on operator precedence.
- Python expressions only contain identifiers, literals and operators.
- **literals**(字面常量)
    - Literals are notations for constant values of some built-in types.
    - String literal, bytes literal, integer, float number, image number
    - E.g. ‘hello’, b‘spam’, 100, 3.14, 1+3j

Immutable

Operators

- Operators are special symbols that perform specific operations on one or more operands (values) and then return a result.
- Python has serveral operators that we can use to perform different mathematical, logical, and boolean operations on data.
- Arithmetic operators
  - It works the same as basic mathematics.
  - +, -, *, /, // floor division, % modulus, ** exponent

Operators

- // floor division
  - 对商向下取整
  - 5//2, 9//4, -7//3, 3.0//2, 3.3//3
- % modulus
  - 取模： a = a%b + b*a//b 对商向下取整
  - 模 a%b 一定是个非负数
  - 8%2, 7%2, -7%3

Operators

- Relational (comparison) operators
  - It performs a comparison between two values. It returns a boolean True or False depending upon the result of the comparison.
  - >, <, ==, !=, >=, <=
- Assignment operators
  - =, +=, -=, *=, /=, %=, //= , **=

Operators

- Logical operators/Boolean operators are useful when checking a condition is true or not. Python has three logical operators. All logical operator returns a boolean value True or False depending on the condition in which it is used.
    - Logical **and**: True if both the operands are True
    - Logical **or**: True if either the operands is True
    - Logical **not**: True if the operand is False

Operators

- Membership operators
  - Python’s membership operators are used to check for membership of objects in sequence, such as string, list, tuple. It checks whether the given value or variable is present in a given sequence. If present, it will return True else False.
  - **in**, **not in**
- Identify operators
  - Use the identify operator to check whether the value of two variables is the same or not. The identify operator compares values according to two variables’ member addresses.
  - **is**, **is not**

Operators

- Bitwise operators
  - In Python, bitwise operators are used to performing bitwise operations on integers. To perform bitwise, we first need to convert integer value to binary (0 and 1) value.
  - The bitwise operator operates on values bit by bit, so it’s called **bitwise**. It always returns the result in decimal format. Python has 6 bitwise operators listed below.
  - & bitwise and, | bitwise or, ^ bitwise xor, ~ bitwise 1’s complement, << bitwise left-shift, >> bitwise right-shift
  - For example, **&** performs logical AND operation on the integer value after converting an integer to a binary value and gives the result as a demical value.

Operators Precedence

- In Python, operator precedence and associativity play an essential role in solving the expression. We must know what the precedence (priority) of that operator is and how they will evaluate down to a single value.
- Operator precedence is used in an expression to determine which operation to perform first.

Precedence level	**Operator**	**Meaning**
1 (Highest)	()	Parenthesis
2	**	Exponent
3	+x, -x ,~x	Unary plus, Unary Minus, Bitwise negation
4	*, /, //, %	Multiplication, Division, Floor division, Modulus
5	+, -	Addition, Subtraction
6	<<, >>	Bitwise shift operator
7	&	Bitwise AND
8	^	Bitwise XOR
9	|	Bitwise OR
10	==, !=, >, >=, <, <=	Comparison
11	is, is not, in, not in	Identity, Membership
12	not	Logical NOT
13	and	Logical AND
14 (Lowest)	or	Logical OR

Statements

- Sentences or Lines
- A statement is an instruction that a Python interpreter can execute.

```
x = 2               Assignment Statement
x = x + 2           Assignment with expression
x += 2
print(x)            print statement
```