BIOS6642

Values, Variables, Expressions

Fuyong Xing

Department of Biostatistics and Informatics

Colorado School of Public Health

University of Colorado Anschutz Medical Campus

Outline

- **Values and types**
- Variables and assignments
- Expressions and operators
- Comments
- Errors

Values and types

- **Value:** One of the basic units of data, like an integer, a floating-point number or string, that a program manipulates.
- **Types:** A category of values.
  - 2: type of integer (int)
  - 3.6: type of floating-point number (float)
  - 'Hello': type of string (str)
- Note that 2,000 is not a legal integer in Python.

Values and types

- **Value:** One of the basic units of data, like an integer, a floating-point number or string, that a program manipulates.
- **Types:** A category of values.
  - 2: type of integer (int)
  - 3.6: type of floating-point number (float)
  - 'Hello': type of string (str)
- Note that 2,000 is not a legal integer in Python.
- Do 2 and '2' have the same values? Why or why not?

Values and types

- In Python, a **string** is a sequence of characters.
- Python recognizes both single quotes (') and double quotes (") as valid ways to delimit a string value.
- The left and right symbols (i.e., the quotes) should be consistent.

```
>>> 'ABC'
'ABC'
>>> "ABC"
'ABC'
>>> 'ABC"
  File "<stdin>", line 1
    'ABC"
        ^
SyntaxError: EOL while scanning string literal
>>> "ABC'
  File "<stdin>", line 1
    "ABC'
        ^
SyntaxError: EOL while scanning string literal
```

Values and types

- Multi-line strings
  - Using the newline character

```
>>> x = 'This is a long string with\nseveral worlds'
>>> print(x)
This is a long string with
several worlds
```

Values and types

- Multi-line strings
  - Using tripe quotes

Codes

```
x = '''
This is a multi-line
    string that goes on
for three lines!
'''
print(x)
```

Output

```
This is a multi-line
    string that goes on
for three lines!
```

Values and types

- Multi-line strings obey indentation and line breaks.

Codes

```
x = '''
    A cube has 8 corners:


           7------8
          /|     /|
         3------4 |
         | |    | |
         | 5----|-6
         |/     |/
         1------2
'''
print(x)
```

Output

```
A cube has 8 corners:


       7------8
      /|     /|
     3------4 |
     | |    | |
     | 5----|-6
     |/     |/
     1------2
```

Values and types

- Type conversion: *int* ⟷ *str*.

```
>>> 4
4
>>> str(4)
'4'
>>> '5'
'5'
>>> int('5')
5
```

- Any integer has a string representation, but not all strings have an integer equivalent.

Values and types

- Type conversion: *float* ⟶ *int*.

```
>>> 28.71
28.71
>>> int(28.71)
28
>>> round(28.71)
29
>>> round(19.47)
19
>>> int(19.47)
19
```

- The *int* function drops the fractional part of floating-point number, while the *round* function produces the integer closest to the original floating-point value.

Outline

- Values and types
- **Variables and assignments**
- Expressions and operators
- Comments
- Errors

Variables and assignments

- A **variable** is a name that refers to a value.
- An **assignment statement** creates a variable and gives it a value (i.e., it binds a viable name to an object).

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897932
```

- The symbol = is the assignment operator.

Variables and assignments

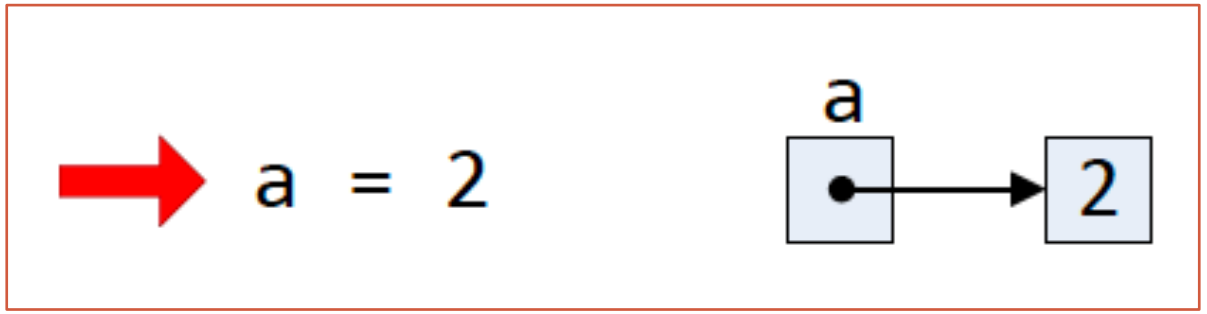
- A **variable** is a name that refers to a value.
- An **assignment statement** creates a variable and gives it a value (i.e., it binds a viable name to an object).

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897932
```

- The symbol = is the assignment operator.
- Is it legal to do *17* = *n* in Python?

Variables and assignments

- The statement a = 2:



- One box represents the variable *a.* The arrow points to another box that contains the value 2. The second box represents a memory location that holds the internal representation of the value 2.

Variables and assignments

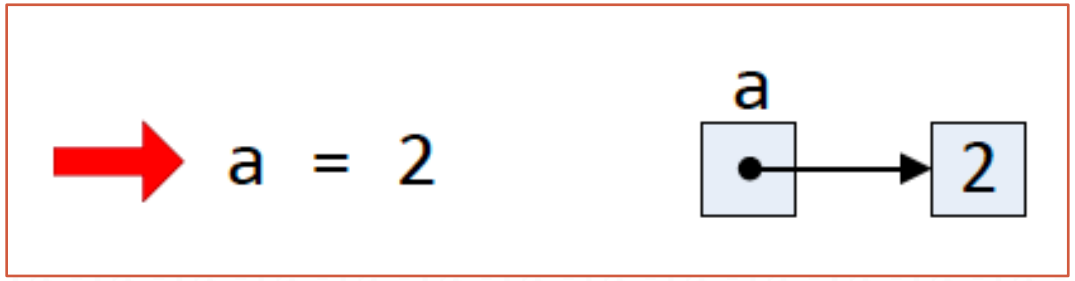
- Consider the following sequence of statements:

```
a = 2
b = 5
a = 3
a = b
b = 7
```

Variables and assignments

- Consider the following sequence of statements:

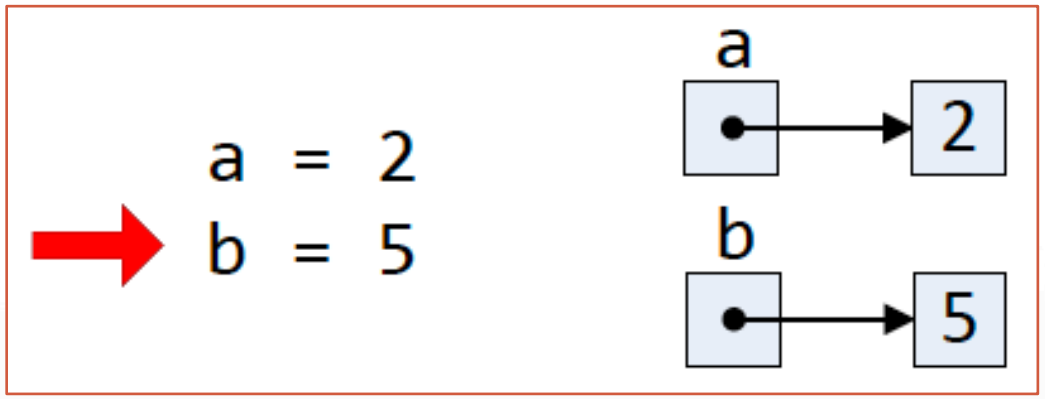
```
a = 2
b = 5
a = 3
a = b
b = 7
```



Variables and assignments

- Consider the following sequence of statements:

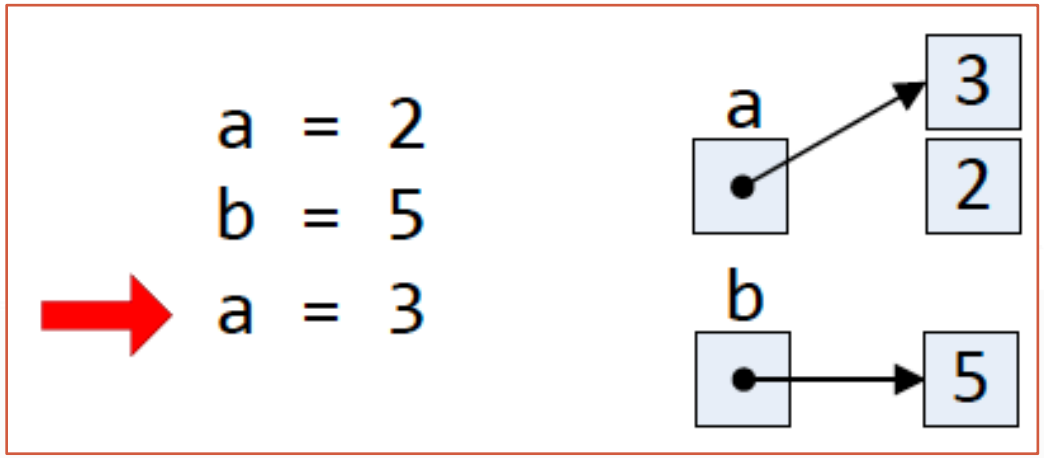
```
a = 2
b = 5
a = 3
a = b
b = 7
```



Variables and assignments

- Consider the following sequence of statements:

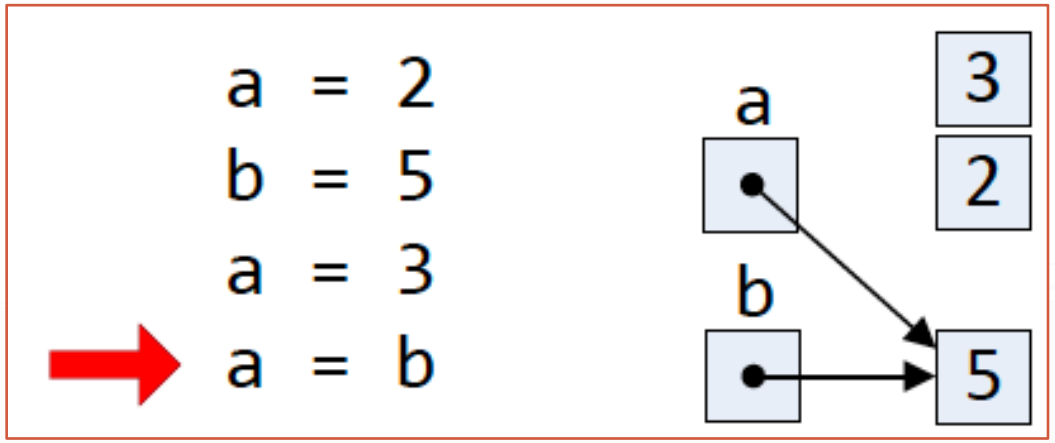
```
a = 2
b = 5
a = 3
a = b
b = 7
```



Variables and assignments

- Consider the following sequence of statements:

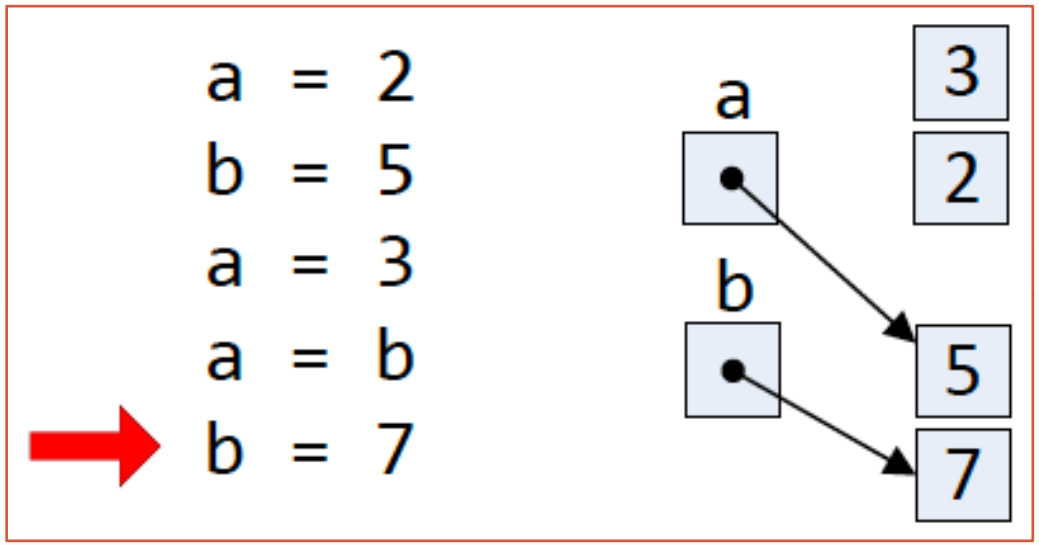
```
a = 2
b = 5
a = 3
a = b
b = 7
```



Variables and assignments

- Consider the following sequence of statements:

```
a = 2
b = 5
a = 3
a = b
b = 7
```



Variables and assignments

- Suppose variables x = 2 and y = 3. How to interchange the values of x and y such that x = 3 and y = 2?
  - Can we use the following code?

```
x = y
y = x
```

Variables and assignments

- Suppose variables x = 2 and y = 3. How to interchange the values of x and y such that x = 3 and y = 2?
  - Can we use the following code?

```
x = y
y = x
```

  - The correct code should be

```
temp = x
x = y
y = temp
```

Variables and assignments

- Suppose variables x = 2 and y = 3. How to interchange the values of x and y such that x = 3 and y = 2?
  - Can we use the following code?

```
x = y
y = x
```

  - The correct code should be

```
temp = x
x = y
y = temp
```

  - We can also use tuple assignment:

```
x, y = y, x
```

Variables and assignments

- A **variable name** is one example of an identifier.
- An **identifier** is a word used to name things.
- An identifier can name a variable, a function, a class, etc.

Variables and assignments

- An identifier has the following form:
  - An identifiers must contain at least one character.
  - The first character of an identifiers must be an alphabetic letter (upper or lower case) or the underscore
  - The remaining characters (if any) may be alphabetic characters (upper or lower case), the underscore, or a digit.
  - No other characters (including spaces) are permitted in identifiers.
  - A reserved word cannot be used as an identifier.

Variables and assignments

- Examples

- `x`
- `x2`
- `total`
- `port_22`

Legal

- `sub-total` (dash is not a legal symbol in an identifier)
- `first entry` (space is not a legal symbol in an identifier)
- `4all` (begins with a digit)
- `*2` (the asterisk is not a legal symbol in an identifier)
- `class` (`class` is a reserved word)

Illegal

Variables and assignments

- Python keywords:

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

Variables and assignments

- A variable name should be well chosen: it should reflect the variable's purpose within the program.
- Good variable names make programs more readable by humans.
- It is conventional to use only lower case for variables names, although it is legal to use uppercase letters.
- Variable names are case sensitive (Actually Python is a case-sensitive language).

Outline

- Values and types
- Variables and assignments
- **Expressions and operators**
- Comments
- Errors

Expressions and operators

- An expression is a combination of values, variables, and operators.
- A value by itself is considered an expression, and so is a variable.

```
>>> 42
42
>>> n
17
>>> n + 25
42
```

Expressions and operators

- An expression is a basic building block of a Python statement.
- Each expression has a value (and a type).
- An expression is evaluated by the interpreter, i.e., determining the value of the expression.
- A literal value like *42* and a variable like *n* are examples of simple expressions. We can use operators to combine values and variables and form more complex expressions.

```
>>> 42
42
>>> n
17
>>> n + 25
42
```

Expressions and operators

- Commonly used Python arithmetic operators

Expression	Meaning
$x + y$	x added to y, if x and y are numbers x concatenated to y, if x and y are strings
$x - y$	x take away y, if x and y are numbers
$x * y$	x times y, if x and y are numbers x concatenated with itself y times, if x is a string and y is an integer y concatenated with itself x times, if y is a string and x is an integer
x / y	x divided by y, if x and y are numbers
$x // y$	Floor of x divided by y, if x and y are numbers
$x \% y$	Remainder of x divided by y, if x and y are numbers
$x ** y$	x raised to y power, if x and y are numbers

Expressions and operators

- What is the type of *sum*?

```
>>> x = 2
>>> y = 3
>>> sum = x + y
```

Expressions and operators

- What is the type of *sum*?

```
>>> x = 2
>>> y = 3
>>> sum = x + y
```

- What is the type of *z*?

```
>>> x = 6
>>> y = 3
>>> z = x / y
```

Expressions and operators

- What is the type of *sum*?

```
>>> x = 2
>>> y = 3.0
>>> sum = x + y
```

Expressions and operators

- What is the type of *sum*?

```
>>> x = 2
>>> y = 3.0
>>> sum = x + y
```

- All arithmetic operators applied to floating-point numbers produce a floating-point result.
- Except in the case of the / operator, arithmetic expressions that involve only integers produce an integer result.

Expressions and operators

- Operator precedence and associativity
- The operators in each row have a higher precedence than the operators below it.

Arity	Operators	Associativity
Binary	**	Right
Unary	+, -	
Binary	*, /, //, %	Left
Binary	+, -	Left
Binary	=	Right

- Parentheses have the highest precedence.

Expressions and operators

- What are the results?

```
>>> 1 + 2 * 3
>>>
>>> (1 + 2) * 3
>>>
>>> 2 ** 3 ** 2
```

Expressions and operators

- What are the results?

```
>>> x = 2
>>> y = 3
>>> x += 1
>>>
>>> x *= x + y
```

Expressions and operators

- Get inputs from the user

```
num1 = int(input('Please enter an integer value: '))
num2 = int(input('Please enter another integer value: '))
print(num1, '+', num2, '=', num1 + num2)
```

Expressions and operators

- Get inputs from the user

```
num1 = int(input('Please enter an integer value: '))
num2 = int(input('Please enter another integer value: '))
print(num1, '+', num2, '=', num1 + num2)
```

- What if 3.4 is typed by the user?

```
>>> num = int(input('Please enter a number: '))
Please enter a number: 3
>>> num
3
>>> num = int(input('Please enter a number: '))
Please enter a number: 3.4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '3.4'
```

Expressions and operators

- String operations

```
>>> x = 'Python'
>>> y = 'programming'
>>> x + y
'Pythonprogramming'
```

```
>>> x = 'Python'
>>> x*3
'PythonPythonPython'
```

Outline

- Values and types
- Variables and assignments
- Expressions and operators
- **Comments**
- Errors

Comments

- Comments can make the codes more readable by human.
- A comment begins with a # symbol.
- Any text contained within comments is ignored by the Python interpreter.
- Both of the following comments are legal.

```
# Compute the average of the values
avg = sum / number
```

```
avg = sum / number        # Compute the average of the values
```

Comments

- Comments are most useful when they document non-obvious features of the code.
- It would be helpful that the comments explain the purpose of a section of code or why the programmers chose to write a section of code the way they did.
- Which one is better?

```
# Assign 0 to the variable named result
result = 0
```

```
# Ensure the variable, result, has a well-defined minimum value
result = 0
```

Outline

- Values and types
- Variables and assignments
- Expressions and operators
- Comments
- **Errors**

Errors

- There are three types of errors in Python: syntax errors, run-time errors and semantic errors.
- **Syntax errors**: An error in a program that makes it impossible to parse (and therefore impossible to interpret).

```
x = 2 # This is syntactically correct
```

```
2 = x # This is not legal
```

Errors

- **Run-time errors**: An error that is detected while the program is running.
- They are also called **exceptions**.

```
x = 2
y = 0
z = x / y       # division by zero
```

Errors

- **Semantic errors**: An error in a program that makes it do something other than what the programmer intended.
- Compared with the other two type of errors, semantic errors might be more difficult to find and fix.

```
>>> print('one half as a percentage: ', 1/2)
one half as a percentage:  0.5
```

Readings

- Think Python, chapter 2

References

- *Think Python: How to Think Like a Computer Scientist*, 2nd edition, 2015, by Allen Downey
- *Python Cookbook*, 3rd edition, by David Beazley and Brian K. Jones, 2013
- *Python for Data Analysis*, 2nd edition, by Wes McKinney, 2018
- *Fundamentals of Python Programming,* by Richard L. Halterman