Chapter 1: Python Basics

Stephen Huang

January 23, 2023

Quotes

“Comprehension of computer code is much easier than writing computer code.”

“We know screw-ups are an essential part of what we do here. That’s why our goal is simple: We just want to screw up as quickly as possible. We want to fail fast. And then we want to fix it.”

— Lee Unkrich, Pixar

“Practice, practice, practice.”

Contents

1. My first Python program
2. Basic Types
3. Assignment Statements
4. Expressions and Precedence
5. Statements and Lines
6. String Basics
7. Input/Output Basics
8. Object Basics
9. Other Stuffs

1. My First Python Program

- The first program in most languages is typically the Hello World program which writes out “hello world.”
- First, we will show you a C++ version and then the Python version.
- Please tell me which one you prefer.

Hello World in C++

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World!\n";
    return 0;
}
```

Hello World in Python

```
print ("Hello World!");
```

Running the program

Python 3.5.2 Shell

IDLE

File Edit Shell Debug Options Window Help

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55)
[MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more infor
mation.
>>>
>>>
>>>
>>>
>>>
>>> print ("Hello World!")
Hello World!
>>>
```

Ln: 8 Col: 0

Running the program

- Type the name of the Python file in command mode.

```
D:\Test>helloworld1.py
Hello World!

D:\Test>helloworld2.py
Hello World!


D:\Test>
```

Running the program

- Click on the program name in the directory

Name	Date modified	Type	Size
helloworld1.py	11/18/2022 8:31 PM	Python File	1 KB
helloworld2.py	11/18/2022 8:32 PM	Python File	1 KB

```
D:\Test>more helloworld1.py
# Name:  Stephen Huang
# This version will pause
#
print("Hello World!")   # In-line comment.

D:\Test>
```

Variables

- Computer can do small tasks very fast.
  - Adding numbers
  - Comparing two numbers
- Most job requires many simple computations to get the result.
- We must use variables to save partially computed results until the whole job is done.
- We may have to <u>reuse</u> some variables to accomplish the job.

Variables

- Programmers generally choose names for their variables that are meaningful—they document what the variable is used for.
- Rules:
  - A variable name can contain both letters and digits, but it can't begin with a digit.
  - The underscore character '_' can appear in a name (treat it as a letter). You will see why later.
  - Python is "case sensitive." Lower case letter ≠ upper case letter.

Examples

- x, y, z
- X, Y, Z
- X1, x123, x1y2
- studentname
- student_name
- studentName
- high_score
- testAverage

- 1st_name
- more@
- Send$
- student-name
- class

Not valid names

Keywords

- The interpreter uses keywords to recognize the program's structure, and they cannot be used as variable names.
- Also called reserved words.

Python 3 has these keywords:

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

2. Basic Types

- Some of the types are built into the Python language:
  - Numeric types
    - Integer (**`int`**) *
    - Floating point numbers (**`float`**) *
    - Complex number (**`complex`**)
  - Boolean (**`bool`**) *
  - String (**`str`**) *

Type Conversion

- **`int`** can be converted into **`float`**.
- **`float`** can be converted into **`int`** (number truncated) subject to some limitation on the size.
- Some strings such as “123” can be converted to numbers too using functions.

Boolean

- These are False:
  - None
  - False
  - zero of any numeric type,
  - any empty sequence,
  - any empty mapping,
  - if a class defines a __bool__() or __len__() method, when that method returns the integer zero or bool value False.
- All other values are considered True.

Python's Typing

- It is not critical for you to understand typing now.
- Python is strongly and dynamically typed.
  - Strong typing means that the type of value doesn't suddenly change. Every change of type requires an <u>explicit</u> conversion.
  - Dynamic typing means that runtime objects (values) have a type, as opposed to static typing where variables have a type.

Typing

- Weak typing does indeed mean that a high percentage of types can be implicitly coerced, attempting to guess what the coder intended.
- Strong typing means that types are not coerced or coerced less.

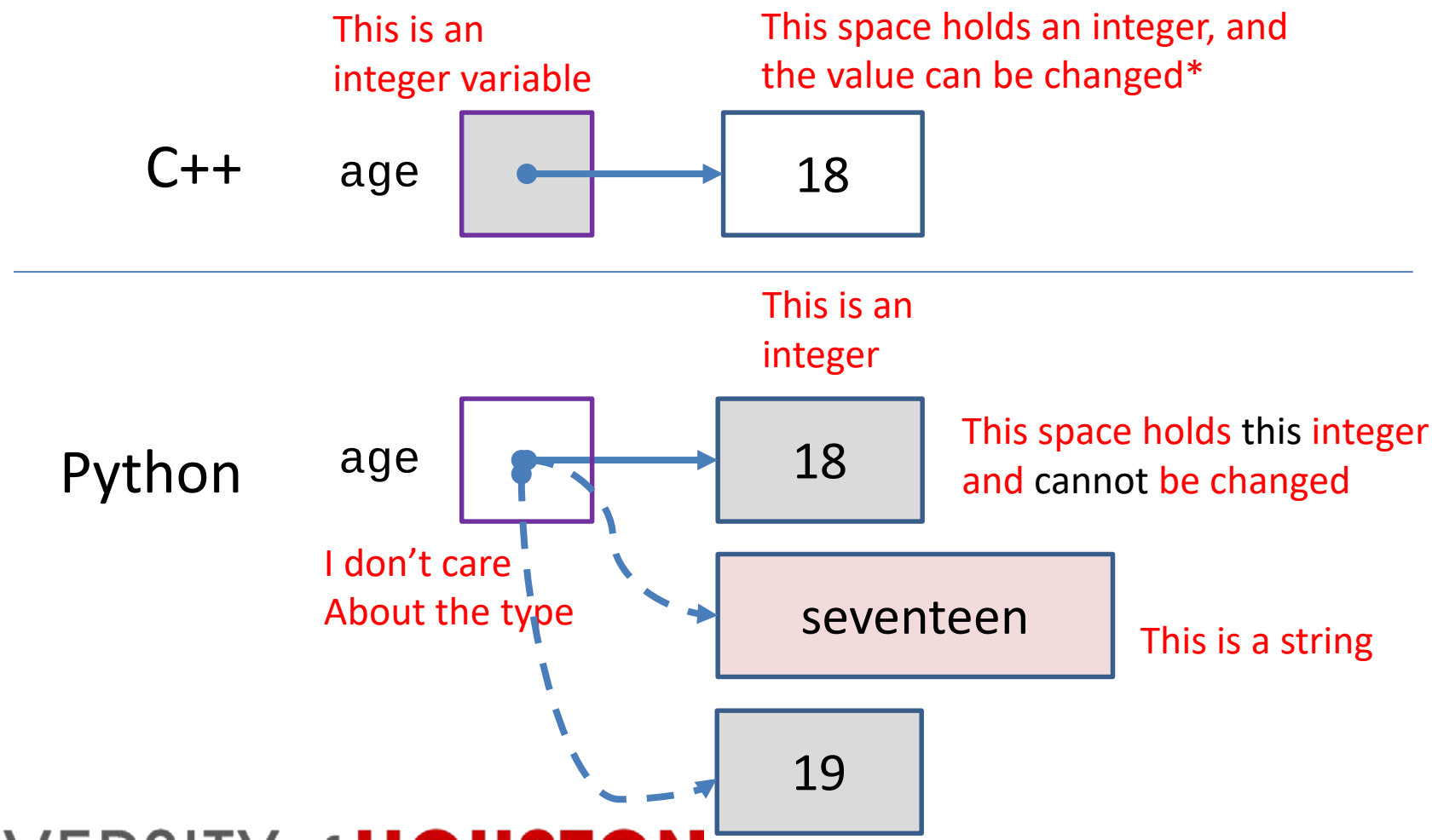
- Static typing means your variables' types are determined at compile time.

Storing a Value

- Unlike C++, Python does not require the user to "declare" a type for a variable.
- Dynamically typed languages (such as Python) allow the type of a variable to change at runtime.
- In contrast, statically typed languages (such as Java or C++) do not allow this once a variable is declared.

A Comparison

- Let's use an integer type as an example.



Typing

- There are times that we want to do type conversion.
  - Cast is explicit.
  - Coerce is implicit.
- Most operators work on values of the same type. What happens if they are not the same?
- Examples:
  - `1.0 + 2  # coercion`
  - `1.0 + float(2)  # casting`

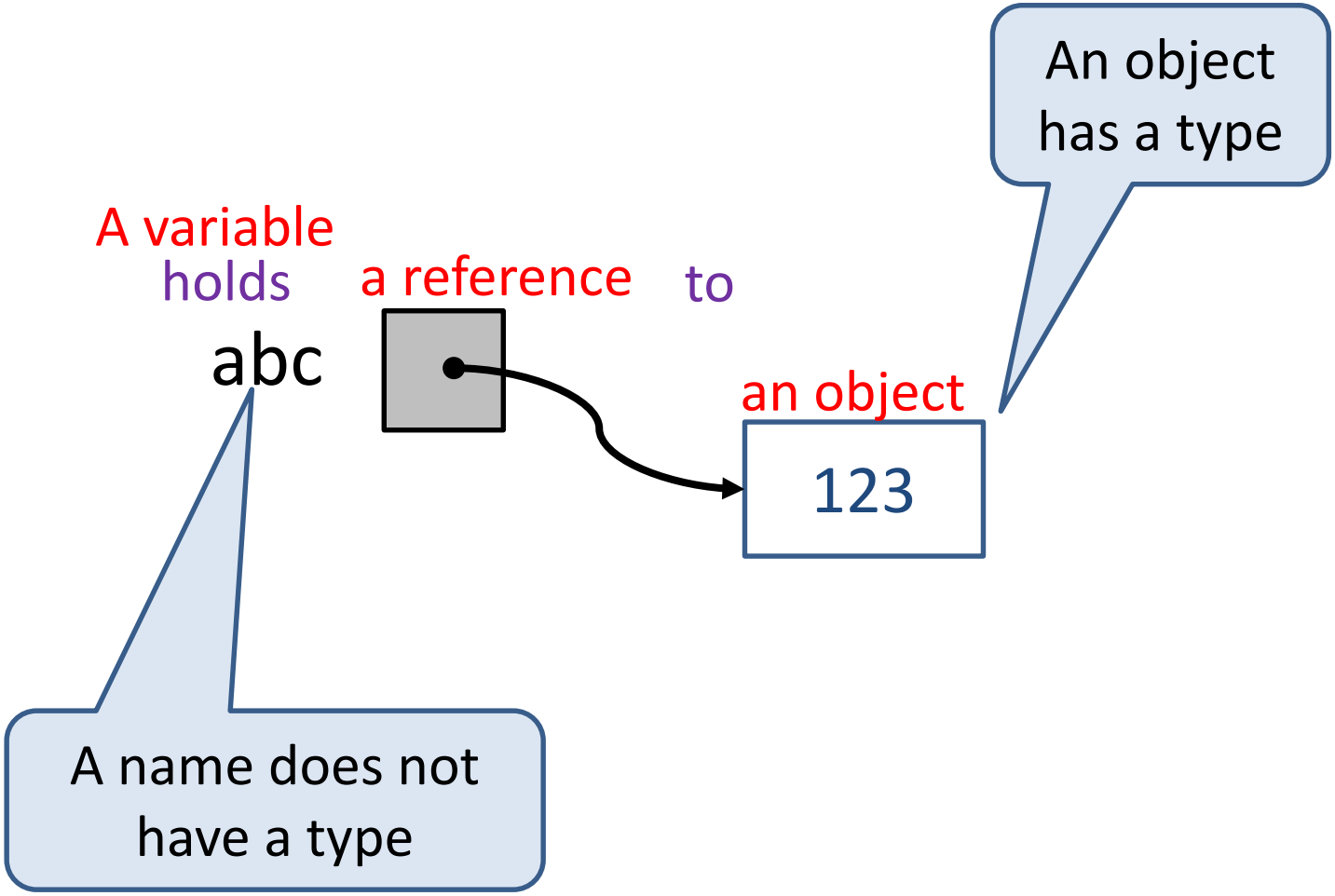
How?

- How do we allow the type of a variable to change?

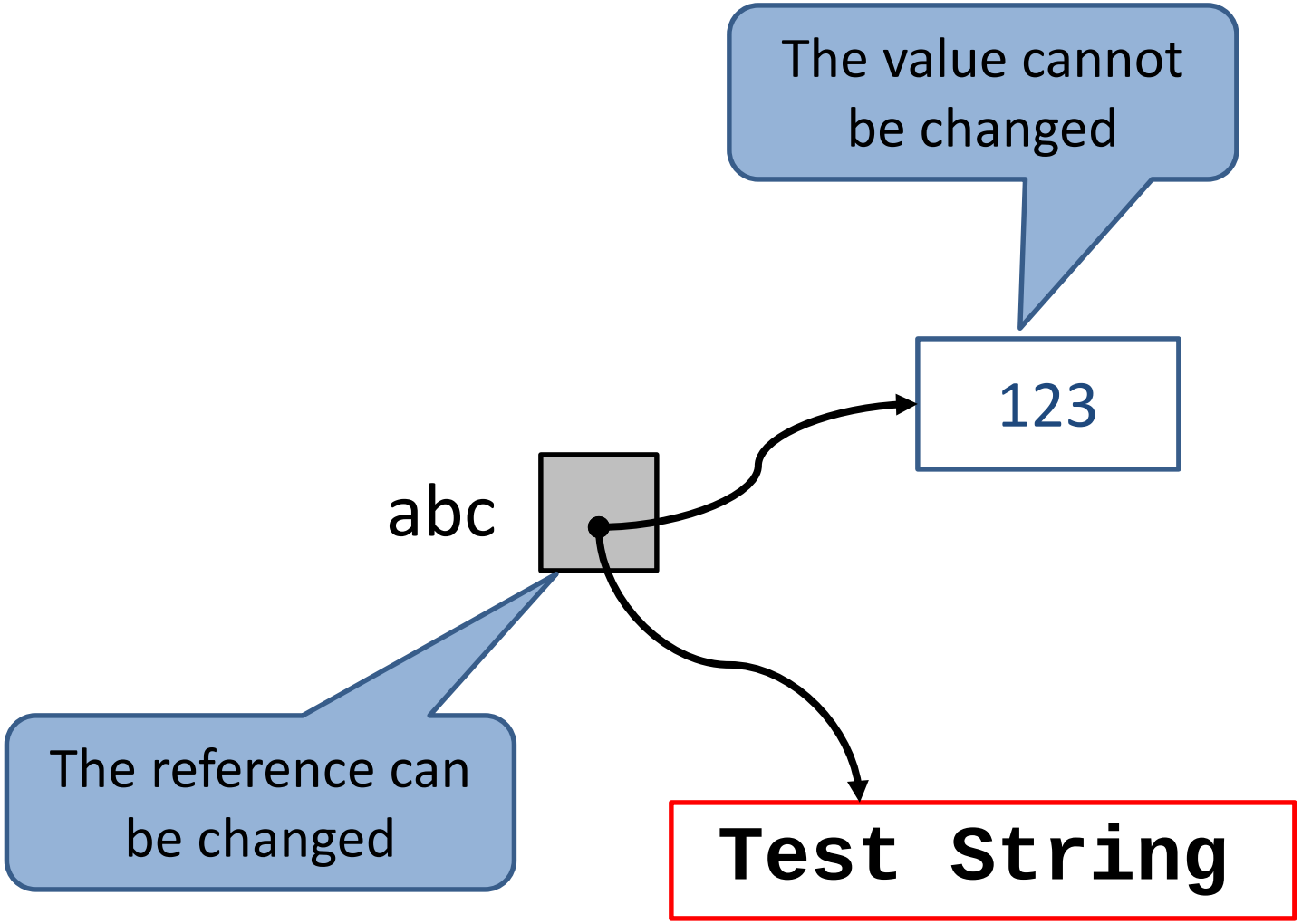
```
a = 1
print (a, type(a))
a = a + 1
a = "Test String"
print (a, type(a))
a = a + 1 # does not work:
```

```
1 <class 'int'>
Test String <class 'str'>
```

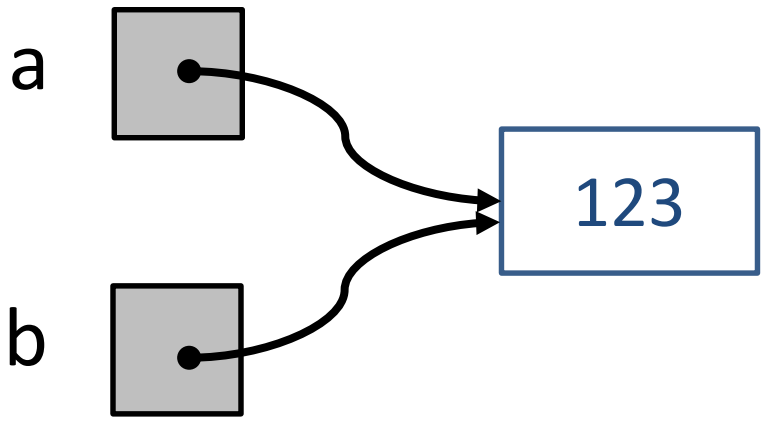
How does it work in Python?



How does it change value?



Two references to an object



3. Assignment Statement

<variable> = <expression>

- A variable is a name that refers to a value.
- An assignment statement creates a new variable and gives it a value.
  - Assignment uses `=`;
  - Comparison uses `==`.
- The value can be the result of an expression.

```
message = 'And now for something different'
n = 17 * 3
pi = 3.141592653589793
```

Executing Assignment

- Long version: Python runs the following two steps:
  - Evaluate the **expression** to produce a **value** (or an object).
    - Ignore the object for now.
    - This value will live at a specific memory address in your computer.
  - Store the value's **memory address** in the **variable**.
    - This step creates a new variable if the current one doesn't already exist, or
    - Updates the value of an existing variable.
- Short version: The variable gets the value.

Multiple Assignments

- x, y = 22, 33

- x, y, z = a, a+1, a+2

- x, y = y, x

Simultaneously

Expressions

- An expression is a combination of values, variables, and operators, resulting in a value.
- Operators are optional, so a value or a variable is considered an expression.
- Commonly used operators:
  - Addition: `+`
  - Subtraction: `-`
  - Multiplication: `*`
  - Division: `/`, `//` (floor division)
  - Remainder: `%` (modulus)
  - Exponentiation: `**`

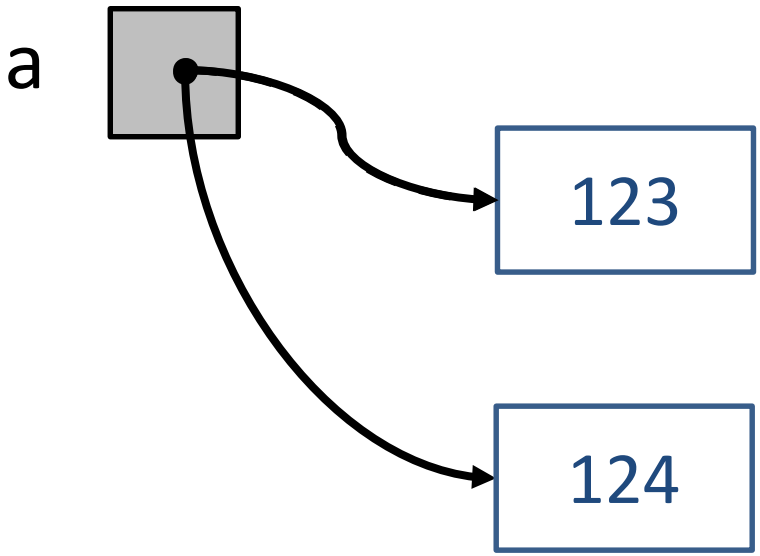
Symbols

- Programming languages typically use many special symbols. However, we have only a limited number of symbols on the keyboard.
  - Solution: use multiple characters such as `**`.
- When using these multi-character symbols, no space or end-of-line separates the characters.
  - `* *` is not the same as `**`.
- There is no left- (“) or right-quote (”). Powerpoint displays them that way. They should be: ' or ".

Assignment Statement

- An assignment statement can assign an expression to a variable.
- General form: Variable = Expression
- Read it as assign the <u>expression value</u> to the variable. Do not read = as equal.
- What is "i = i + 1"?
  - Add 1 to the (old) value of i and assign the result value to i.
  - Mathematically, "i = i+1" does not make sense.

Example



Examples

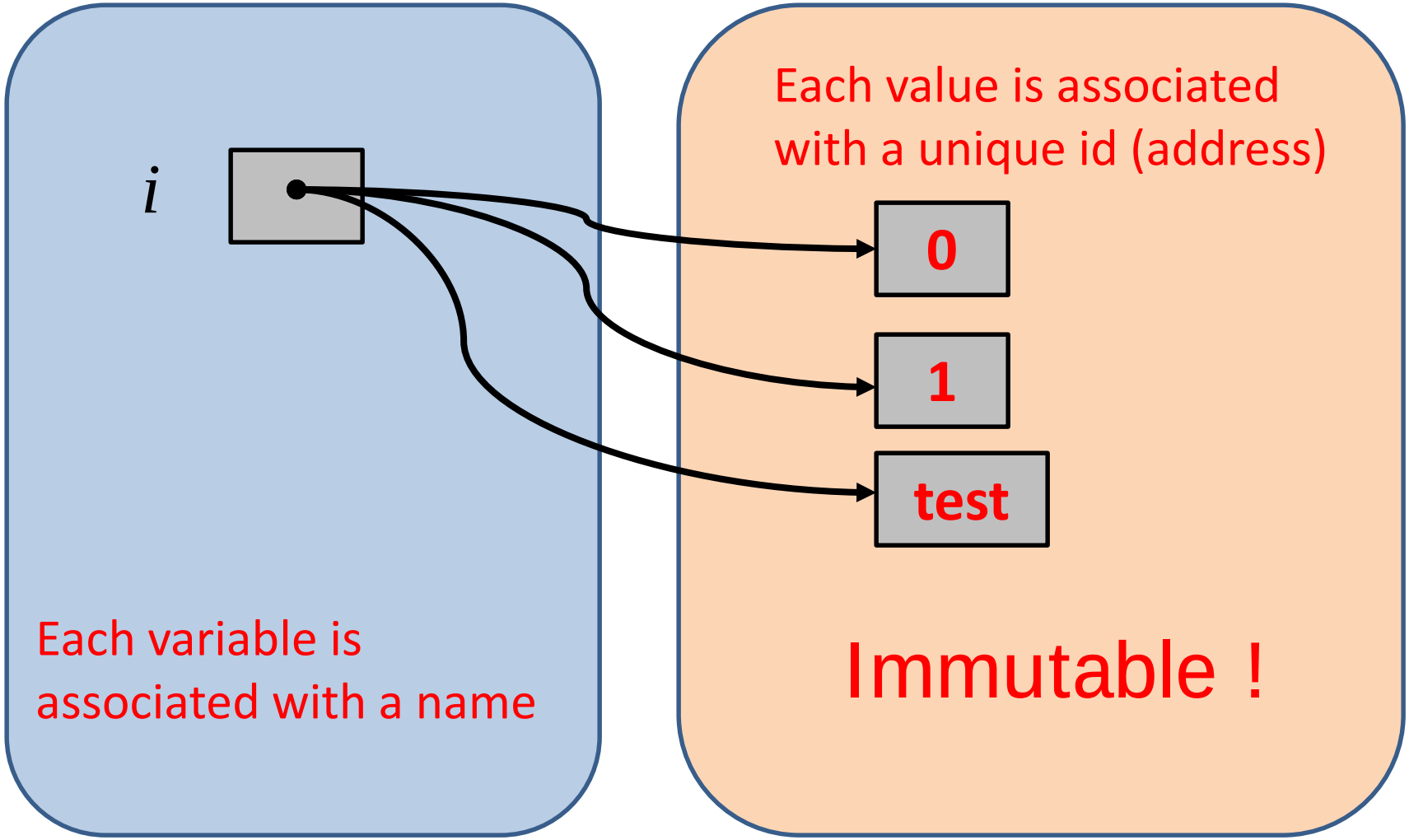
Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

```
>>>
>>> year = 2017
>>> year
2017
>>> 2016
2016
>>> year - 1
2016
>>> year + 1
2018
>>>
>>> |
```

Ln: 42 Col: 4

What Happened?



Example

- Computing simple interest

Python 3.6.0 Shell

File Edit Shell Debug Options Window Help

```
on.
>>>
>>>
>>> principal = 1000
>>> interestRate = 1 # percent
>>> timePeriod = 5
>>> finalValue = principal * (1+interestRate/100*timePeriod)
>>> finalValue
1050.0
>>>
```

Ln: 10 Col: 0

Shell vs. Scrpt

Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

```
>>>
>>> miles = 26.2
>>> miles * 1.61
42.182
>>>
```

Nothing

Test - [C:\Users\Stephen\Test] - ...\test002 - PyCharm Edu 3.0.1

File Edit View Navigate Code Help

Test > test002

test002

```
1  miles = 26.2
2  miles * 1.61
3
```

Run test002

```
Process finished with exit code 0
```

4: Run Python Console

Statement seems to have no effect 3:1 n/a UTF-8

Script Mode

Test - [C:\Users\Stephen\Test] - ...\test002 - PyCharm Edu 3.0.1

File Edit View Navigate Code Help

test002

Test test002

1: Project

test002

```
1  miles = 26.2
2  print (miles * 1.61)
3
```

Run test002

```
42.182

Process finished with exit code 0
```

4: Run Python Console

4:1 n/a UTF-8

4. Expressions and Precedence

- When an expression contains more than one operator, the evaluation of the expression depends on the precedence of operators.
- Operators with higher precedence (priority) will be executed before lower precedence operators.
- Python follows mathematical convention.

Order can be changed

- Parentheses have the highest precedence and can force an expression to evaluate in the order you want since expressions in parentheses are evaluated first.
- Exponentiation has the next highest precedence.
- Multiplication and Division have higher precedence than Addition and Subtraction.
- Operators with the same precedence are evaluated from <u>left to right</u> (except for exponentiation).

Examples

Python 3.6.0 Shell

File Edit Shell Debug Options Window Help

```
>>>
>>> 512/4/2
64.0
>>> (512/4)/2
64.0
>>> 512/(4/2)
256.0
>>>
>>> |
```

Ln: 26 Col: 4

Left to right

Python 3.6.0 Shell

File Edit Shell Debug Options Window Help

```
>>>
>>> 2**3**2
512
>>> (2**3)**2
64
>>> 2**(3**2)
512
>>>
>>> |
```

Ln: 19 Col: 4

Right to left

Parentheses

- You can always use parentheses to make the meaning of an expression evident, even though they are not necessary.

```
x * b  +  c / (d -  e % f)

(x * b) + (c / (d – (e % f)))
```

Operator Precedence

Table 4.2 Order of Evaluation (Highest to Lowest)

Operator	Name
`(...), [...], {...}`	Tuple, list, and dictionary creation
`´...´`	String conversion
`s[i], s[i:j]`	Indexing and slicing
`s.attr`	Attributes
`f(...)`	Function calls
`+x, -x, ~x`	Unary operators

Operator Precedence

`x ** y`	Power (right associative)
`x * y, x / y, x // y, x % y`	Multiplication, division, floor division, modulo
`x + y, x - y`	Addition, subtraction
`x << y, x >> y`	Bit-shifting
`x & y`	Bitwise and
`x ^ y`	Bitwise exclusive or
`x | y`	Bitwise or

Operator Precedence

`x < y, x <= y, x > y, x >= y,`	Comparison, identity, and sequence membership tests
`x == y, x != y`	
`x <> y`	
`x is y, x is not y`	
`x in s, x not in s`	
`not x`	Logical negation
`x and y`	Logical and
`x or y`	Logical or

The is operator

- Equal (==) is not identical (is).
- The == operator is True when the values of two operands are equal.
- The <u>is</u> operator is True if the two variables point to the same object.

5. Statements and Lines

- Logically, a program consists of one or more statements.
- Physically, a program consists of many characters divided into lines.
- Good programming practice: No more than one statement per line whenever possible.

Statements

- A statement is a unit of code that has an effect, like creating a variable or displaying a value.
  - An assignment statement is an example.
  - “pass” is a statement that does nothing.
- A statement must follow the specific syntax of the language. The following few lectures will be about syntax.

Statement and Line

- What if a statement is longer than a line?
  - You can also use a backslash (\) to indicate that a statement continues onto the following line.
- A blank line signals the end of a “block” (for-loop, for example).

Logical vs. Physical Lines

	1 Physical Line	2+ Physical Lines
1 Statement	1-1	Join (\)
	Almost Always	Sometimes
2+ Statements	Separator (;)	Many-Many
	Avoid it	Avoid, Avoid, Avoid

Join

- Explicit Join: “\”
- Implicit Join: Since `[]` `()` `{}` are always used in pairs, a pair of brackets indicate a statement even if it is crossing line boundary.
- Note: it odes not apply to quote-unquote.

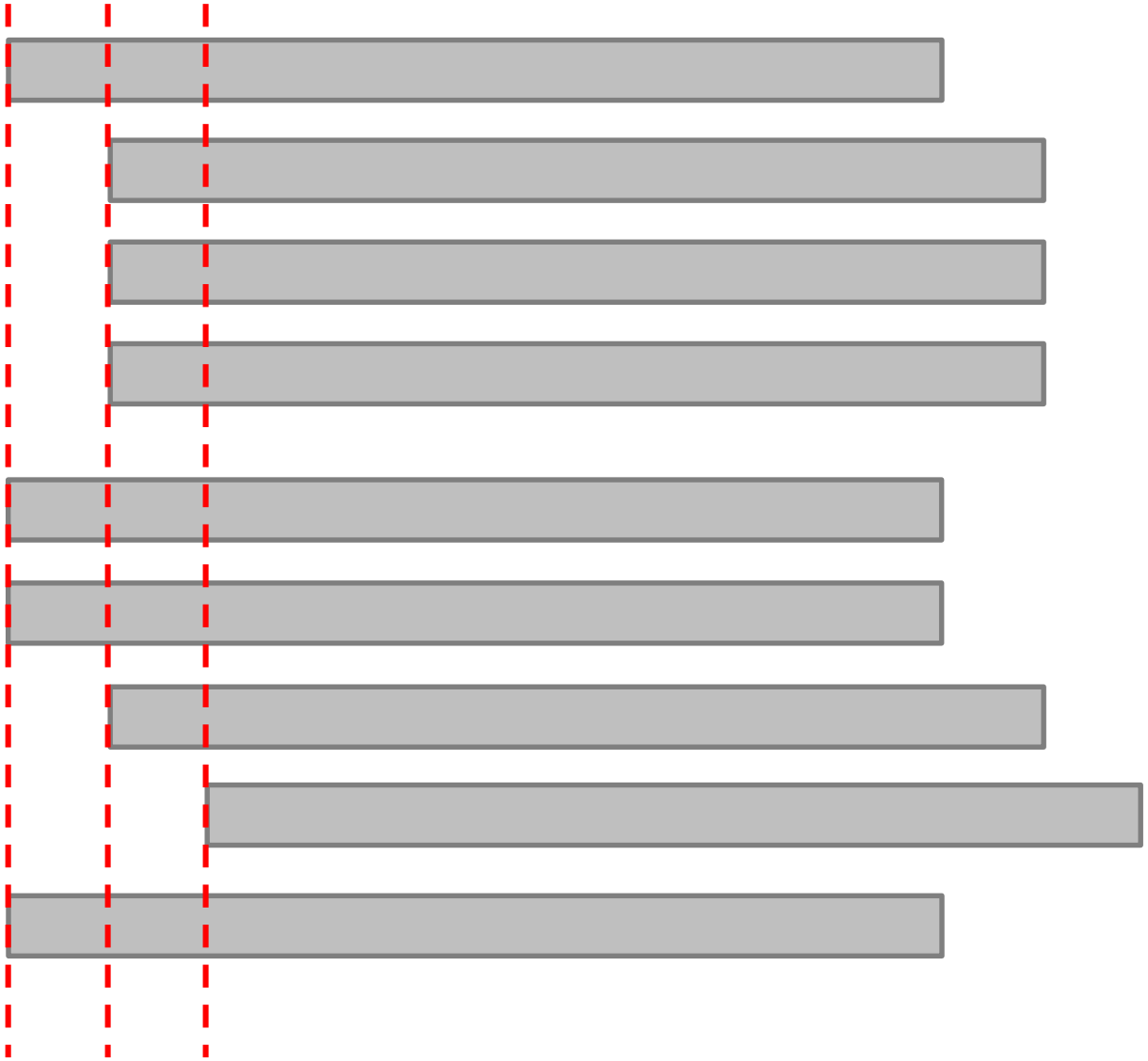
Whitespace

- You get nothing (in a space) in print when you print certain characters in any programming language. Therefore, they are called whitespace characters.
- The most common ones:
  - blank character
  - newline (end-of-line)
  - tab
- Whitespace is meaningful in python: especially indentation and placement of newlines.

Whitespace

- Use a newline to end a line of code (almost always). Use join when it is necessary to go to the following line.
- Python uses indentation to mark block; no braces **{ }** are needed.
- The first line with less indentation is outside of the prior block
- The first line with more indentation starts a nested block.

Indentation



Quick Start

- We are going to show some simple syntax and rules in Python so that you can understand simple statements in the examples.
- They include:
  - Comments
  - String Basics
  - Input/Output Basics
  - Object Basics

6. String Basics

- A string consists of 0 or more characters.
- Strings can be enclosed in single quotes ('...') or double quotes ("...") with the same result.
- Since (single or double) quote has a special meaning, we have to be careful in including them in a <u>quoted</u> string.
  - It is okay to have a single quote inside a double-quoted string and vice versa.
- Triple quotes ('''). You cannot find ''' on your keyboard. It's three consecutive single quotes.

Escape Character

- "\" can be used to <u>escape</u> quotes.

```
a = 'spam eggs'
b = "spam eggs"
c = "Don't do that."
d = 'Don\'t do that.'
e = '"Yes," he said.'
f = "\\n"
```

```
spam eggs
spam eggs
Don't do that.
Don't do that.
"Yes," he said.
\n
2
```

Display Strings

- In the interactive interpreter, the output string is enclosed in quotes and special characters are escaped with backslashes.
- The print() function produces a more readable output.

```
>>> s ='First line\nSecond line'
>>> s
'First line\nSecond line'
>>> print (s)
First line
Second line
>>> |
```

Examples

```
>>>
>>> s = 'single \' double " inside single.'
>>> t = "single \  double \" inside double."
>>> s
'single \' double " inside single.'
>>> t
'single \\  double " inside double.'
>>>
```

```
>>> s = '\\\\'
>>> len (s)
2
>>> s
'\\\\'
>>> print(s)
\\
>>>
```

Strings

- In general, you can't perform mathematical operations on strings.

Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

```
>>>
>>> 'third' * 'a charm'
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    'third' * 'a charm'
TypeError: can't multiply sequence by non-int of type 'str'
>>>
>>> '2' - '1'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    '2' - '1'
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>>
```

Ln: 27 Col: 4

Exceptions

- Two exceptions: + and *:
  - '+' is concatenation
  - '*' is repetition

Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

```
>>>
>>> 'star' + 'wars'
'starwars'
>>>
>>> 'Tora! ' * 3
'Tora! Tora! Tora! '
>>>
>>>
```

Ln: 40 Col: 0

Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

```
>>>
>>>
>>> '3' * 4
'3333'
>>>
>>> '4' * 3
'444'
>>>
```

Ln: 47 Col: 4

Multi-line Strings

- There are ways to have a long string across multiple lines.

```
quotation = "Well written code \
is its own best documentation."
print (quotation)
quotation = "Well written code " \
            "is its own best documentation."
print (quotation)
quotation = "Well written code " + \
            "is its own best documentation."
print (quotation)
quotation = ("Well written code " +
            "is its own best documentation.")
print (quotation, "\n", type(quotation))
quotation = ("Well written code "
            "is its own best documentation.")
print (quotation, "\n", type(quotation))
```

Default Print Parameters

- You can print multiple values in one print() call. The values will be separated by a separator string (the default value is a space).
- The default ending string is a newline (eoln) character. So, the second print always started with a newline.
- You can change the separator string and the ending string.

```
print (a, b, c, sep=' ', end='\n')
```

7. Input/Output Basics

- Input syntax:

```
variable = input(prompt)
```

- Prompt is a string to be displayed.
- Example:

```
person = input('Enter your name: ')
print('Hello', person)
```

Hello Steve!

```
1  person = input('Enter your name: ')
2  print('Hello',   person)
3  print('Hello'  + person)
4  print('Hello ' + person)
5  print('Hello',   person, '!')
6  print('Hello',   person, end='!')
```

Run input-001

```
Enter your name: Steve
Hello Steve
HelloSteve
Hello Steve
Hello Steve !
Hello Steve!
```

ASCII Input

- All inputs are in ASCII form, i. e., they are characters.
- If you want a number, you will have to convert (cast) or evaluate it.

```
xString = input("Enter a number: ")
x = int(xString)
```

or

```
x = int(input("Enter a number: "))
```

Simple Output

- `print(value, ..., sep=' ', end='\n')`
  - sep and end must be at the end of the parameter list.
  - The **sep** string separates the value list. (multiple)
  - The **end** string terminates the value list. (only one)
- The print was a statement in Python 2.
- It becomes a function in Python 3.
- Python is not “backward compatible.”

Example

```python
print (1, 2, 3)
print (1, 2, 3, sep=" ", end="\n")
print (1, 2, 3, end=" ", sep = "**")
print (4, 5, 6, sep="",  end="\n")
print (7, 8, 9, sep=" #\n", end=" !\n\n")
```

Run print-001

```
1 2 3
1 2 3
1**2**3 456
7 #
8 #
9 !
```

String Methods

- Text Processing Services:
  - `str.isalnum()`
  - `str.isalpha()`
  - `str.isdecimal()`
  - `str.isdigit()`
  - `str.islower(), str.isupper()`
  - `str.lower(), str.upper()`
  - `str.isnumeric()`
  - `str.isspace()`
- Your IDE can help.

8. Object Basics

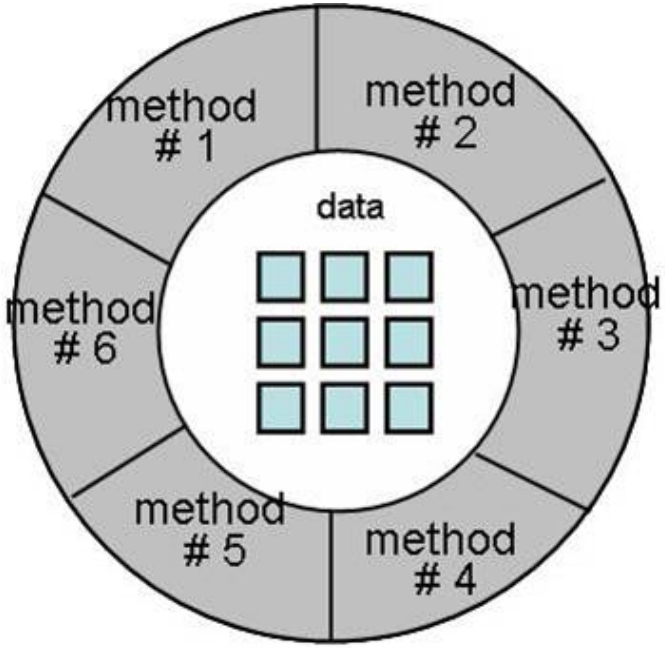
- Python is an Object-Oriented Programming (OOP) language.
- OOP provides a way of organizing programs that is similar to the way people think.
- We do not plan to discuss OOP in detail in 1306.
- We will learn just enough to continue with other chapters.

Objects

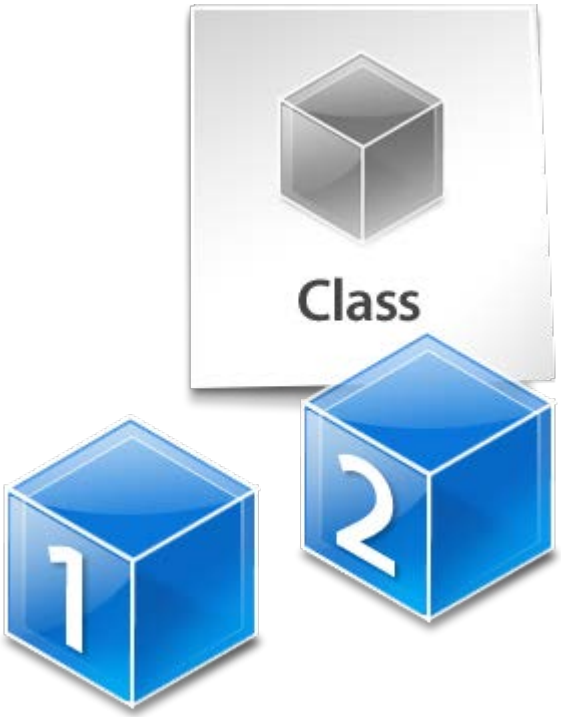
- An object is a collection of data and functions.
- To distinguish traditional data from the data contained in an object, we call the data in an object the object's attributes.
- To distinguish traditional functions from the functions contained in an object, we call the functions in an object the object's methods.
- In python, everything is an object. Data is an object, and so is a function.

Classes

- A class statement provides a blueprint for creating objects.
- An object's type corresponds to its class.



Objects are instances of a Class



Access

- An object is a collection of attributes and methods.
- To access an object's attributes or methods, one writes the object followed by the access attribute operator, i.e., a dot (.), followed by the desired attribute or method.

```
<object>.<attribute>
<object>.<method>(<params>)
```

- Keep in mind that the dot is an operator and cannot be part of an identifier.

An Example

```
class Student:
    name = "Jane Doe"
    age = 18
    stu_class = "Freshman"
    def display(self):
        print("Name          =", self.name)
        print("Age           =", self.age)
        print("Student Class  =", self.stu_class)

anderson = Student()
anderson.name = "Robert Anderson"
anderson.display()
```

Equal?

- Each object in python has
  - an identity,
  - a type,
  - a value.
- To compare the object identity
  - `a is b`
- To compare the type
  - `type(a) is type(b)`
- To compare the object values
  - `a == b # 2 diff objs, same value`

10. Other Stuffs

- The `pass` statement does nothing.
- It can be used when a statement is required syntactically, but the program requires no action.
- Indentation is Python's way of grouping statements.

Other Stuffs

- Python allows multiple assignments as the following examples show.

```
def show (a, b, c):
    print ("a = ", a, " b = ", b, " c = ", c)
a, b, c = 9, 99, 999
show (a, b, c)
b = c = d = 100
show (a, b, c)
a, b = b, a
show (a, b, c)
```

Run assign-001

```
a =  9  b =  99  c =  999
a =  9  b =  100  c =  100
a =  100  b =  9  c =  100
```

Comments

- Comments are any text as notes for the reader of the program.
  - explain assumptions and limitations,
  - explain important decisions, details
  - explain problems you're trying to solve

Comments

- It is nice to add notes to your programs to explain what they are doing in natural language.
- These notes are called comments, and they start with the # symbol.
- Everything from the # to the end of the line is ignored — does not affect the program's execution.

Proper way to use comments

- This comment is redundant with the code and useless:

```
v = 5 # assign 5 to v
```

- This comment contains useful information that is not in the code:

```
v = 5 # velocity in meters/second
```

- A better way:

```
velocity_mps = 5 # meters/second
```

- “in-line” comments

Example

Test - [C:\Users\Stephen\Test] - ...\test002 - PyCharm Edu 3.0.1

File Edit View Navigate Code Help

```
1  distance = 100 # miles
2  time = 3.5     # seconds
3  #
4  # velocity in miles per second
5  #
6  velocity = distance / time
7  print (velocity)
8
```

Multiline Comments

- Triple quotes (single or double)

Python

```python
"""
If I really hate pressing `enter` and
typing all those hash marks, I could
just do this instead
"""
```

Python

```python
# -*- coding: utf-8 -*-
"""A module-level docstring

Notice the comment above the docstring specifying the encoding.
Docstrings do appear in the bytecode, so you can access this through
the ``__doc__`` attribute. This is also what you'll see if you call
help() on a module or any other Python object.
"""
```

Documentation String

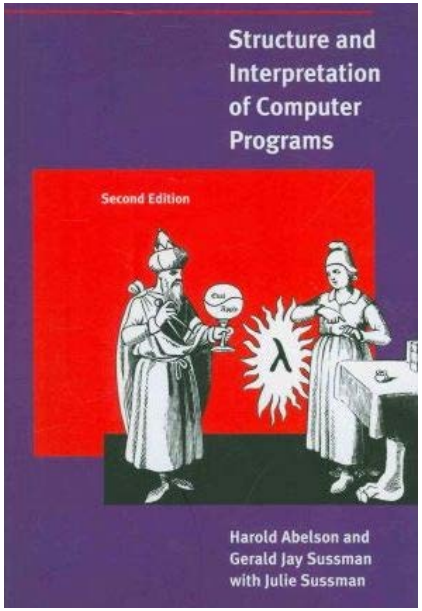
- Include a “docstring” as the first line of any new function or class you define.
- The development environment, debugger, and other tools may use the info.
  - `print(my_function.__doc__)`
  - `help(my_function)`

```
def my_function(x, y):
    """This is the docstring. This
    function does blah blah blah."""
    # The code would go here...
```

When to comment?

- "Programs must be written for people to read, and only incidentally for machines to execute."

  - Harold Abelson and Gerald Jay Sussman, 1985.



An Example in Perl

You may very well know that

```
$string= join('',reverse(split('',$string)));
```

reverses your string, but how hard is it to insert

```
# Reverse the string
```

into your program?

Code vs. Comment

- Code Tells You How, Comments Tell You Why

- "Code can only tell you how the program works; comments can tell you why it works,"
  - Jeff Atwood, 2006.

CODING HORROR
programming and human factors

Debugging

- Three kinds of errors can occur in a program
  - Syntax error: “Syntax” refers to the structure of a program and the rules about that structure.
  - Runtime error: so-called because the error does not appear until after the program has started running.
  - Semantic error: The third type of error is “semantic,” which is related to the meaning.

Forward Reference

- Logically, most languages do not allow you to "use" something that you have not "defined."
  - You cannot print a value x before giving it a value.
- There are a few exceptions. Recursion is an example.