PROGRAMMING
Lecture 05

Hanbat National University
Dept. of Computer Engineering
Changbeom Choi

OUTLINE

Review: what we have done

Objects: Values and types

Variables

Operators and operands

Expressions

Reading assignment

- Chapter 3 of the **textbook**
- The lecture note for **cs1media**

Download two image files, **images and photos**

REVIEW

Characteristics of Python

Instruction set

Arithmetic and logical operations

- +, -, *, /, and **
- and, or, not

$\left.\begin{array}{l} \\ \\ \end{array}\right\}$ for defining expressions

Assignment

Conditionals

Iterations

Input/output

No pointers

No explicit declarations

What we have learned

Through 2D robot control we learned:

- conditionals: `if`, `if~else`, and `if~elif~else`
- iterations
  - for-loops
  - while-loops
- assignment, e.g., hubo = Robot()
- functions

Picked up the main constructs for programming.

OBJECTS: VALUES AND TYPES

Programs work with **data.** **Every piece of data** in a Python program is called an **object** , e.g.,

3, 5.7, “Smith”, True, …	simple
a digital photograph , hubo, …	complex

A **value** itself is an **object.**

Every object has a **type**. The **type** determines **what you can do with an object.**

Python Zoo
Imagine there is a **zoo** inside your Python interpreter. Every time an **object** is created, an **animal** is born. **What an animal can do** depends on **the kind of animal: birds can fly, fish can swim, elephants can lift weights**, etc. When an animal is **no longer** used, it **dies**(disappears).

How to create objects?

Simple objects: by writing them

Numbers

```
    integer: 13, -5
    float: 3.14159265
    complex number: 3 + 6j
```

Strings(a piece of text)

```
  “Programming is wonderful”
  “Programming is great”
  “The instructor said: ‘Well done!’ and smile”
```

Booleans(truth values)

```
     True or False
```

Complex objects

User-defined objects: by calling functions that create them

```
from cs1robots import *
hubo = Robot()

from cs1media import *
load_picture("photos/geowi.jpg")
```

Data structures (objects composed of another objects):

by writing them

Tuples

(1, 3, 5, 7, 9)

("red", "green", "blue")

(777, "a lucky number")

Lists

Dictionary

$\Big\}$ to be discussed later

Tuples

```
position = (3.0,  7.2,  5.7)
Instructors = (“Joseph S. Shin”, “Chang B. Choi”)
```

A **tuple** is a **single object** of **type tuple**:

```
>>> print position, type(position)
(3.0, -7.2, 5.7) <type 'tuple'>
```

We can **unpack** tuples:

```
x, y, z = position
```

Object types: The **type** of an object determines **what the object can do** or **what you can do with the object.** For instance, you can add two numbers, but you cannot add two robots.

Type inquires

```
>>>type(3)
<Type ‘int’>
>>>type(3.145)
<Type ‘float’>
>>>type(“Welcome”)
<Type ‘str’>
```

```
>>>type(3 + 5j)
<Type ‘complex’>
>>>type(True)
<Type ‘bool’>
```

```
>>> from cs1robots import *
>>> type(Robot())
<class 'cs1robots.Robot'>

>>>from cs1media import *
>>> type( load_picture(“photos/geowi.jpg") )
<class 'cs1media.Picture'>

>>> type( (3, -1.5, 7) )
>>><type 'tuple'>
```

VARIABLES

A **variable** is a **name** that refers to an **object**(or a **value**).
An **assignment** statement is used to **define** a **variable**:

```
message = “Welcome“
n = 17
from cs1robots import *
hubo = Robot()

pi = 3.1415926535897931
finished = True

from cs1media import *
img = load_picture(“photos/geowi.jpg")
```

In the Python zoo, the name is a sign board on the animal’s cage.

Rules for **variables** and **function names:**

A name consists of **letters**, **digits**, and the **underscore,**
The **first character** of a name is a **letter**.
The name **cannot be a keyword** such as def, if, else, or while.
Upper case and **lower** case are **different**: Pi is not the same as pi.

Good:

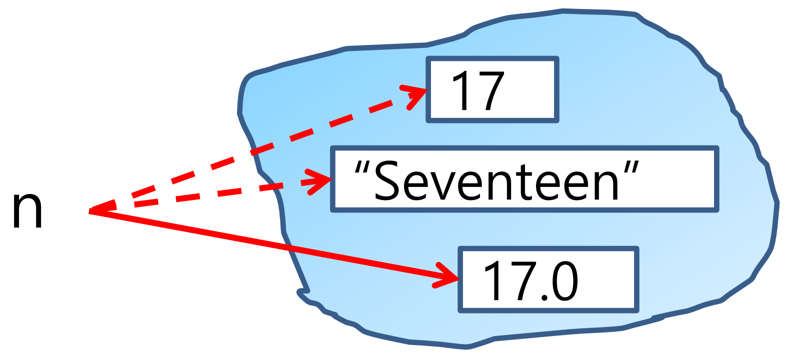
```
msg = “Programming is fantastic“
ba13 = 13.0
```

Bad:

```
more@ = "illegal character“
13a = 13.0
def = “Definition”
```

The **same name** can be assigned to **different objects** (of **different types**) in a program, e.g.,

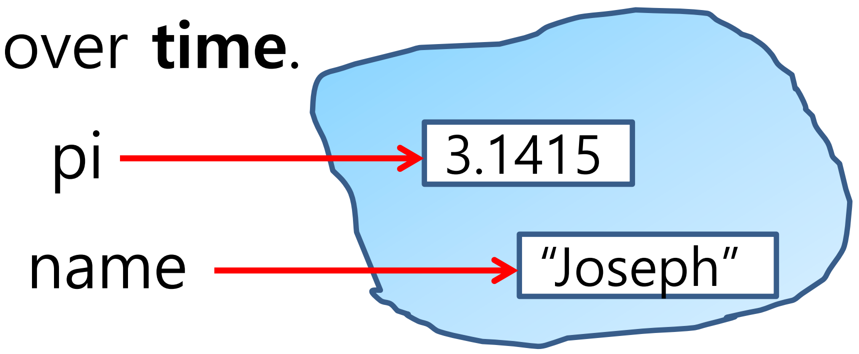
```
n = 17
n = "Seventeen”
n = 17.0
```



In the Python zoo, this means that the sign board is moved from one animal to a different animal.

The **object** binding to a **variable** is called the **value** of the **variable.** The **value** can change over **time**.

```
pi   = 3.1415
name = "Joseph"
```



To indicate that a variable is **empty**, we use the **special object** None (of type NoneType):

```
m = None
```

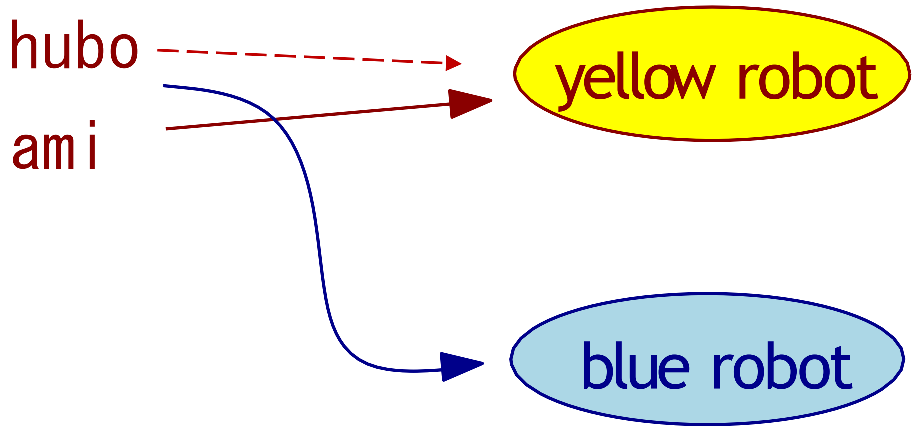
What objects can do depends on the **type of object**: a bird can fly, a fish can swim. Objects provide **methods** to perform these actions. The **methods** of an **object** are used through **dot-syntax:**

```
>>> b = "banana"
>>> print (b.upper())
BANANA
>>>from cs1robots import *
>>> hubo = Robot()
>>> hubo.move()
>>> hubo.turn_left()
>>>from cs1media import *
>>>img = load_picture(“images/pikachu.png”)
>>> print (img.size())
(274, 256)
>>> img.show()
```

```
hubo = Robot("yellow”)
hubo.move()
ami = hubo
```

The same object may have more than one name!

```
hubo = Robot("blue")
hubo.move()
ami.turn_left()
ami.move()
```



OPERATORS AND OPERANDS

Arithmetic operators are special symbols that represent **computations** such as **+, -, *, /, %,** and ******. **Operands** are the **values** to which an operator is applied.

```
>>> 2 ** 16        a ** b = a^b
65536
>>>15.3 + 3.0
18.3
>>>7 % 5
2
>>>7 // 5
1
>>>7 / 5
1.4
```

EXPRESSIONS AND STATEMENTS

Expressions

An **expression** is a combination of **objects**, **variables**, **operators**, and **function calls**:

```
3.0 * (2 ** 15 - 12 / 4) + 4 ** 3
```

The **operators** have **precedence** as in mathematics:

1. exponentiation `**`
2. multiplication and division `*` , `/`, `%`
3. addition and subtraction `+`, `-`

When in doubt, use parentheses!

How to represent $\frac{a}{2pi}$? Which ones are right?

a/2*pi   a/(2*pi)   a/2/pi

The **operators** + and * can be used for **strings**:

```
>>> "Hello " + “Programming”
‘Hello Programming’

>>> “Programming " * 8
‘Programming Programming Programming … Programming”
```

Repeating 8 times!

Relational operators ==, !=, >, <, <=, and >= are used to compare objects. The results are **Boolean values, True** or **False.** A **Boolean expression** is an expression whose **value** is of **type bool**. They are used in if and while statements.

```
>>>27 == 14
False
>>> 3.14 != 3.14
False

>>> 3.14 >= 3.14
True
>>> “Cheong” < “Choe”
True
>>> "3" == 3
False
```

```
x = 9
if x == 3 ** 2 :
     print ( “x is a perfect square”)

if x % 2 != 0:
   print  (“x is odd”)
```

The keywords **not, and,** and **or** are **logical operators**:

not True $\longrightarrow$ Flase

not False $\longrightarrow$ True

False and False	$\longrightarrow$ False	False or False	$\longrightarrow$ False
False and True	$\longrightarrow$ False	False or True	$\longrightarrow$ True
True and False	$\longrightarrow$ False	True or False	$\longrightarrow$ True
True and True	$\longrightarrow$ True	True or True	$\longrightarrow$ True

```
x = 5.0
y = 6.0
z = 7.0

if x < y and y < z:
    print (“z is the largest one.”)
if y < x or y < z:
    print (“ y may not be the least one.”)
if not z >= 6.0:
    print (“z is not the largest one.”)
```

STATEMENTS(INSTRUCTIONS)

conditionals: if, if~else, and if~elif ~else

iteration

- for-loops
- while-loops

assignments a = b

input/output

(functions)

Review: for-loops

for **variable** in range(n):

    block of statements

The **block** of **statements**(instructions) are executed **n times**.
While performing the block, **variable** changes **from 0 to n-1**.
Starting from 0, it is incremented by one at each iteration
to reach n-1.

```
for i in range(4):
    print(i, end =“ ”)
```

What does this short code do?

It prints 0 1 2 3

```
For i in range(7):
    print ("*“ * (i + 1))
```

What does this short code do ?

```
*
**
***
****
*****
******
*******
```