Fall 2018

CS101 – Variables and Basic Data Types

Lecture 3

School of Computing, KAIST

Interpreter vs compiler

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

Roadmap

Last week we learned

- Conditionals and `while` Loops

This week we will learn

- Objects
- Types
- Variables
- Methods
- Tuples

Objects

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

Objects can be very small (the number 3) or very large (a digital photograph).

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

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

Making objects

You can create objects as follows:

Numbers: Simply write them:

```
13
3.14159265
-5
3 + 6j
```

Complex number

Strings: (a piece of text)

Write text between quotation marks (" and ' are both okay):

```
"CS101 is wonderful"
'The instructor said: "Well done!" and smiled'
```

Booleans: (truth values)

Write **`True`** or **`False`**.

Making more objects

Complicated objects are made by calling functions that create them:

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

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

A **tuple** object is an object that contains other objects.

To create a tuple, write objects separated by commas (usually in parenthesis):

```
(3, 2.5, 7)
("red", "yellow", "green")
(20100001, "Hong Gildong")
```

Different animals: Types

Every object has a **type**. The type determines what the object can do, and what you can do with the object. For instance, you can add two numbers, but you cannot add two robots.

The Python interpreter can tell you the type of an object:

```
>>> type(3)
<class 'int'>
>>> type(3.1415)
<class 'float'>
>>> type("CS101 is fantastic")
<class 'str'>
>>> type(3 + 7j)
<class 'complex'>
>>> type(True)
<class 'bool'>
```

Integer number: **int**

Floating point number: **float**

String: **str**

Complex number: **complex**

Boolean: **bool**

More types

Types of more complicated objects:

```
>>> type(Robot())
<class 'cs1robots.Robot'>
>>> type( (3, -1.5, 7) )
<class 'tuple'>
>>> type( load_picture("geowi.jpg") )
<class 'cs1media.Picture'>
```

Names

Objects can be given a **name**:

```
message = "CS101 is fantastic"

n = 17
hubo = Robot()
pi = 3.1415926535897931
finished = True
img = load_picture("geowi.jpg")
```

We call a statement like `n = 17`an **assignment**, because the **name** `n` is **assigned** to the object `17`.

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

Variable names

The rules for variable and function names:

- A name consists of letters, digits, and the underscore _.
- The first character of a name should not be a digit.
- The name should not 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:

- my_message = "CS101 is fantastic"
- a13 = 13.0

Bad:

- more@ = "illegal character"
- 13a = 13.0
- def = "Definition 1"

Variables

Names are often called **variables**, because the meaning of a name is variable: the same name can be assigned to different objects within a program:

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

The object assigned to a name is called the **value** of the variable. The value can be changed over time.

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

```
n = None
```

Methods

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**:

```
>>>  hubo = Robot()
>>>  hubo.move()
>>>  hubo.turn_left()

>>>  img = load_picture("geowi.jpg")
>>>  print(img.size())       # width and height in pixels
(58, 50)
>>>  img.show()     # display the image

>>>  b = "banana"
>>>  print(b.upper())
BANANA
```

Operators

For numbers, we use the operators +, −, *, /, //,%, and **.

a ** b= a^b

```
>>> 2**16
65536
```

Remainder after division

```
>>> 7 % 3
1
```

//is integer division (division without fractional part):

```
>>> 13.0 // 4.0
3.0
>>> 9 / 7
1.2857142857142858
```

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!

e.g., $\frac{a}{2\pi}$ is **not** `a/2*pi`

Use `a/(2*pi)`or `a/2/pi`.

All operators also work for complex numbers.

String expressions

The operators + and * can be used for strings:

```
>>> "Hello" + "CS101"
'HelloCS101'

>>> "CS101 " * 8
'CS101 CS101 CS101 CS101 CS101 CS101 CS101 CS101 '
```

Boolean expressions

A **boolean expression** is an expression whose value has type **bool**. They are used in `if` and `while` statements.

The operators ==, !=, >, <, <=and >=return boolean values.

```
>>> 3 < 5
True
>>> 27 == 14
False
>>> 3.14 != 3.14
False
>>> 3.14 >= 3.14
True
>>> "Cheong" < "Choe"  T
rue
>>> "3" == 3
False
```

Equality – don't confuse with =

Logical operators

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

```
not True  == False
not False == True

False and False == False
False and True == False
True and False == False
True and True == True

False or False == False
False or True == True
True or False == True
True or True == True
```

Careful: If the second operand is not needed, Python does not even compute its value.

Tuples

A tuple is an object that contains other objects:

```
>>> position = (3.14, -5, 7.5)
>>> profs = ("Ko", "Kim", "Myaeng", "Choi", "Choi")
```

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

```
>>> print(position, type(position))
(3.14, -5, 7.5) <class 'tuple'>
```

We can “unpack” tuples:

```
>>> x, y, z = position
>>> print(x)
3.14
```

Packing and unpacking in one line:

```
>>> a, b = ("aa", "bb")
>>> a, b = b, a
>>> print(b)
aa
```

Colors

Colors are often represented as a tuple with three elements that specify the intensity of red, green, and blue light:

```
red = (255, 0, 0)
blue = (0, 0, 255)
white = (255, 255, 255)
black = (0, 0, 0)
yellow = (255, 255, 0)
purple = (128, 0, 128)

from cs1media import *
img = create_picture(100, 100, purple)
img.show()
img.set_pixels(yellow)
img.show()
```

Colors

A digital image of width w and height h is a rectangular matrix with h rows and w columns:

0, 0	1, 0	2, 0	3, 0	4, 0
0, 1	1, 1	2, 1	3, 1	4, 1
0, 2	1, 2	2, 2	3, 2	4, 2

We access pixels using their x and y coordinates.
x is between 0 and w−1, y is between 0 and h−1.

```
>>> img.get(250, 188)
(101, 104, 51)
>>> img.set(250, 188, (255, 0, 0))
```

red, green, blue triple

For loops

A for-loop assigns integer values to a variable:

```
>>> for i in range(4):
...   print(i)
0
1
2
3
```

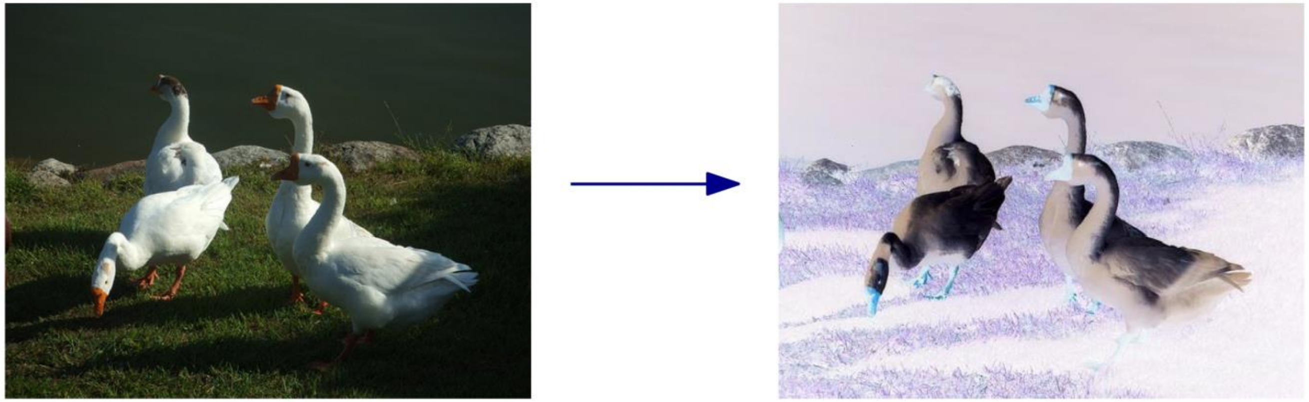
```
>>> for i in range(7):
...   print ("*" * i)

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

Negative of a photo

```
from cs1media import *

img = load_picture("../photos/geowi.jpg")
w, h = img.size()
for y in range(h):
    for x in range(w):
     r, g, b = img.get(x, y)
     r, g, b = 255 - r, 255 - g, 255 - b
     img.set(x, y, (r, g, b))
     img.show()
```



Black & white photo

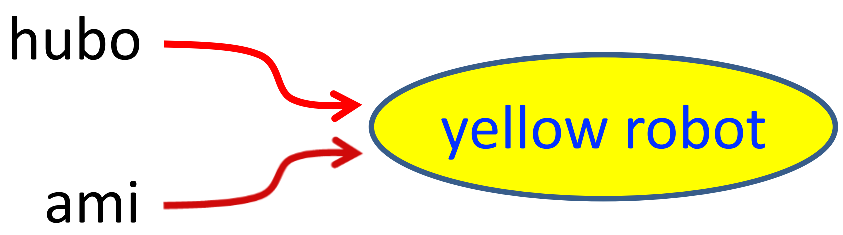
```
from cs1media import *
threshold = 100
white = (255, 255, 255)
black = (0, 0, 0)

img = load_picture("../photos/yuna1.jpg")
w, h = img.size()
for y in range(h):
  for x in range(w):
    r, g, b = img.get(x, y)
    v = (r + g + b) // 3    # average of r,g,b
    if v > threshold:
      img.set(x, y, white)
    else:
      img.set(x, y, black)
img.show()
```

Objects with two names

The same object can have more than one name:

```
hubo = Robot("yellow") hubo.mo
ve()
ami = hubo
ami.turn_left() hubo.mov
e()
```

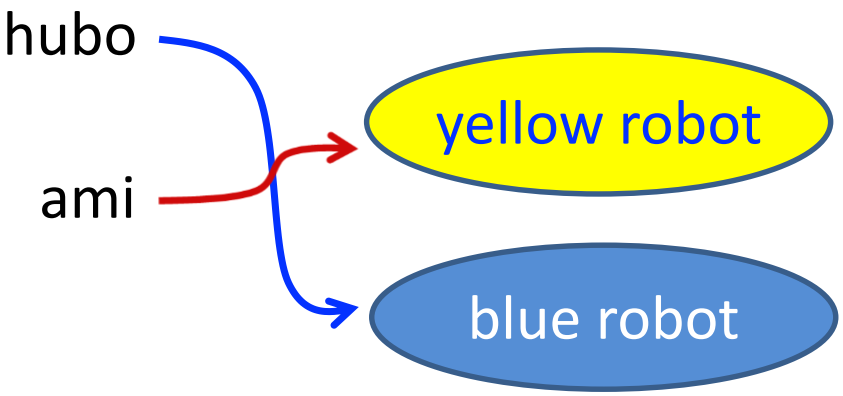


Objects with two names

The same object can have more than one name:

```
hubo = Robot("yellow")  hubo.mo
ve()
ami = hubo
ami.turn_left()  hubo.mo
ve()

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



Questions?