Week 1

Review

Special thanks to Scott Shawcroft, Ryan Tucker, and Paul Beck for their work on these slides.

Python!

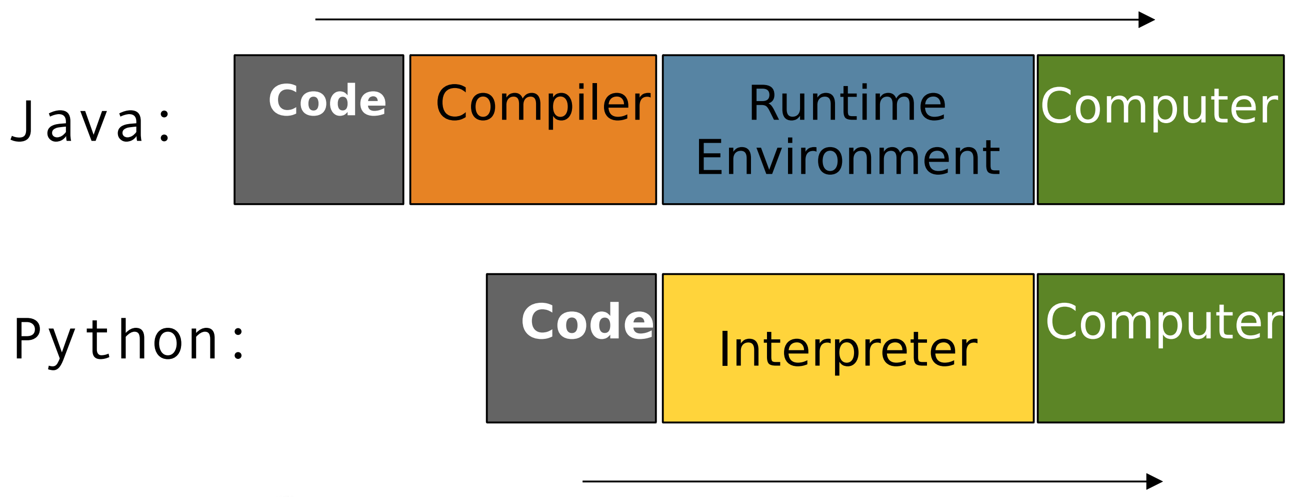
- Created in 1991 by Guido van Rossum (now at Google)
  - Named for Monty Python

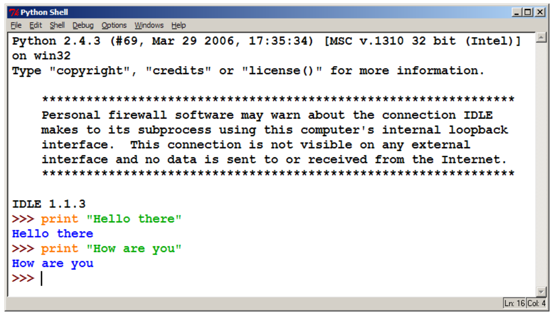
- Useful as a **scripting language**
  - **script**: A small program meant for one-time use
  - Targeted towards small to medium sized projects

- Used by:
  - Google, Yahoo!, Youtube
  - Many Linux distributions
  - Games and apps (e.g. Eve Online)

Interpreted Languages

- **interpreted**
  - Not compiled like Java
  - Code is written and then directly executed by an **interpreter**
  - Type commands into interpreter and see immediate results





```
Python 2.4.3 (#69, Mar 29 2006, 17:35:34) [MSC v.1310 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.1.3
>>> print "Hello there"
Hello there
>>> print "How are you"
How are you
>>> |
```

The `print` Statement

`print("`**`text`**`”)`
`print()` (a blank line)

- Escape sequences such as `\"` are the same as in Java
- Strings can also start/end with `'`

swallows.py

```
print("Hello, world!”)
print()
print("Suppose two swallows \"carry\" it together.”)
Print('African or "European" swallows?’)
```

Comments

# comment text (one line)

swallows2.py

```
# Suzy Student, CSE 142, Fall 2097
# This program prints important messages.
Print("Hello, world!”)
Print()                       # blank line
Print("Suppose two swallows \"carry\" it together.”)
Print('African or "European" swallows?’)
```

Expressions

- Arithmetic is very similar to Java
  - Operators: `+ - * / %` (plus `**` for exponentiation)
  - Precedence: `()` before `**` before `* / %` before `+ -`
  - Integers vs. real numbers

```
>>> 1 + 1
2
>>> 1 + 3 * 4 - 2
11
>>> 7 / 2
3
>>> 7.0 / 2
3.5
```

Variables and Types

- Declaring: same syntax as assignment; no type is written
- Types: Looser than Java
  - Variables can change types as a program is running
- Operators: no ++ or --

Java

```
int x = 2;
x++;
System.out.println(x);

x = x * 8;
System.out.println(x);

double d = 3.2;
d = d / 2;
System.out.println(d);
```

Python

```
x = 2
x = x + 1
print(x)

x = x * 8
print(x)

d = 3.2
d = d / 2
print(d)
```

Value	Java type	Python
42	int	int
3.14	double	float
"ni!"	String	str

String Multiplication

- Python strings can be multiplied by an integer.
  - Result: many copies of the string concatenated together

```
>>> "hello" * 3
"hellohellohello"

>>> 10 * "yo “
yo yo yo yo yo yo yo yo yo yo

>>> 2 * 3 * "4”
444444
```

String Concatenation

- Integers and strings cannot be concatenated in Python.

  Workarounds:
  - `str(value)` - converts a value into a string
  - `print value, value` - prints value twice, separated by space

```
>>> x = 4
>>> "Thou shalt not count to " + x + "."
TypeError: cannot concatenate 'str' and 'int' objects

>>> "Thou shalt not count to " + str(x) + "."
Thou shalt not count to 4.

>>> x + 1, "is out of the question."
5 is out of the question.
```

The for Loop

```
for name in range([min, ] max [, step]):
    statements
```

- Repeats for values **min** (inclusive) to max (exclusive)
  - **min** and **step** are optional (default **min** 0, **step** 1)

```
>>> for i in range(4):
...     print(i)
0
1
2
3
>>> for i in range(2, 5):
...     print(i)
2
3
4
>>> for i in range(15, 0, -5):
...     print(i)
15 10 5
```

Functions

- **Function**: Equivalent to a static method in Java.

```
def name():
    statement
    statement
    ...
    statement
```

hello2.py

```
# Prints a helpful message.
def hello():
    print("Hello, world!”)
    print("How are you?”)

# main (calls hello twice)
hello()
hello()
```

  - 'main' code (not an actual method) appears below functions
  - Statements inside a function must be indented

Parameters

def **name**(**parameter**, **parameter**, ..., **parameter**):
statements

– Parameters are declared by writing their names (no types)

```
>>> def print_many(word, n):
...     for i in range(n):
...         print(word)

>>> print_many("hello", 4)
hello
hello
hello
hello
```

Default Parameter Values

```
def name(parameter=value, ..., parameter=value):
    statements
```

- Can make parameter(s) optional by specifying a default value

```
>>> def print_many(word, n=1):
...     for i in range(n):
...         print(word)

>>> print_many("shrubbery")
shrubbery
>>> print_many("shrubbery", 4)
shrubbery
shrubbery
shrubbery
shrubbery
```

python™

Returning Values

```
def name(parameters):
    statements
    ...
    return value
```

```
>>> def ftoc(temp):
...     tempc = 5.0 / 9.0 * (temp - 32)
...     return tempc

>>> ftoc(98.6)
37.0
```

Math commands

```
from math import *
```

Function name	Description
`ceil(`**value**`)`	rounds up
`cos(`**value**`)`	cosine, in radians
`degrees(`**value**`)`	convert radians to degrees
`floor(`**value**`)`	rounds down
`log(`**value**`,` **base**`)`	logarithm in any base
`log10(`**value**`)`	logarithm, base 10
`max(`**value1, value2, ...**`)`	largest of two (or more) values
`min(`**value1, value2, ...**`)`	smallest of two (or more) values
`radians(`**value**`)`	convert degrees to radians
`round(`**value**`)`	nearest whole number
`sin(`**value**`)`	sine, in radians
`sqrt(`**value**`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

String Methods

Java	Python
`length`	`len(`**`str`**`)`
`startsWith, endsWith`	`startswith, endswith`
`toLowerCase, toUpperCase`	`upper, lower, isupper, islower, capitalize, swapcase`
`indexOf`	`find`
`trim`	`strip`

```
>>> name = "Martin Douglas Stepp"
>>> name.upper()
'MARTIN DOUGLAS STEPP'
>>> name.lower().startswith("martin")
True
>>> len(name)
20
```

input

`input` : Reads a string from the user's keyboard.

- reads and returns an entire line of input

```
>>> name = input("Howdy. What's yer name? ")
Howdy. What's yer name? Paris Hilton

>>> name
'Paris Hilton'
```

- to read a number, cast the result of `raw_input` to an `int`

if/else

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```

- Example:

```
gpa = input("What is your GPA? ")
if gpa > 3.5:
    print("You have qualified for the honor roll.”)
elif gpa > 2.0:
    print("Welcome to Mars University!”)
else:
    print("Your application is denied.”)
```

if ... in

```
if value in sequence:
    statements
```

- The sequence can be a range, string, tuple, or list
- Examples:

```
x = 3
if x in range(0, 10):
    print("x is between 0 and 9”)

name = raw_input("What is your name? ")
name = name.lower()
if name[0] in "aeiou":
    print("Your name starts with a vowel!”)
```

Logical Operators

Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

Operator	Example	Result
`and`	`(2 == 3) and (-1 < 5)`	`False`
`or`	`(2 == 3) or (-1 < 5)`	`True`
`not`	`not (2 == 3)`	`True`

while Loops

```
while test:
    statements
```

```
>>> n = 91
>>> factor = 2        # find first factor of n

>>> while n % factor != 0:
...     factor += 1
...

>>> factor
7
```

bool

- Python's logic type, equivalent to `boolean` in Java
  - `True` and `False` start with capital letters

```
>>> 5 < 10
True

>>> b = 5 < 10
>>> b
True

>>> if b:
...     print("The bool value is true")
...
The bool value is true

>>> b = not b
>>> b
False
```

Random Numbers

`from random import *`

`randint(`**min, max**`)`

- returns a random integer in range [**min**, **max**] inclusive

`choice(`**sequence**`)`

- returns a randomly chosen value from the given sequence
  - the sequence can be a range, a string, ...

```
>>> from random import *
>>> randint(1, 5)
2
>>> randint(1, 5)
5
>>> choice(range(4, 20, 2))
16
>>> choice("hello")
'e'
```

Strings

index	0	1	2	3	4	5	6	7
or	-8	-7	-6	-5	-4	-3	-2	-1
character	P	.		D	i	d	d	y

- Accessing character(s):

  variable [**index**]

  variable [**index1**:**index2**]

  - **index2** is exclusive
  - **index1** or **index2** can be omitted (end of string)

```
>>> name = "P. Diddy"
>>> name[0]
'P'
>>> name[7]
'y'
>>> name[-1]
'y'
>>> name[3:6]
'Did'
>>> name[3:]
'Diddy'
>>> name[:-2]
'P. Did'
```

Tuple

tuple_name = (**value**, **value**, **...**, **value**)

- A way of "packing" multiple values into one variable

```
>>> x = 3
>>> y = -5
>>> p = (x, y, 42)
>>> p
(3, -5, 42)
```

name, **name**, **...**, **name** = **tuple_name**

- "unpacking" a tuple's contents into multiple variables

```
>>> a, b, c = p
>>> a
3
>>> b
-5
>>> c
42
```

Tuple as Parameter/Return

def **name**((**name**, **name**, **...**, **name**), **...**):
statements

- Declares tuple as a parameter by naming each of its pieces

```
>>> def slope((x1, y1), (x2, y2)):
...     return (y2 - y1) / (x2 - x1)

>>> p1 = (2, 5)
>>> p2 = (4, 11)
>>> slope(p1, p2)
3
```

return (**name**, **name**, **...**, **name**)

```
>>> def roll2():
...     die1 = randint(1, 6)
...     die2 = randint(1, 6)
...     return (die1, die2)

>>> d1, d2 = roll2()
```

Reading Files

name = open("**filename**")
- opens the given file for reading, and returns a file object

name.read() – file's entire contents as a string

```
>>> f = open("hours.txt")
>>> f.read()
'123 Susan 12.5 8.1 7.6 3.2\n
456 Brad 4.0 11.6 6.5 2.7 12\n
789 Jenn 8.0 8.0 8.0 8.0 7.5\n'
```

Line-based File Processing

name.`readline()` – next line from file as a string

- Returns an empty string if there are no more lines in the file

name.`readlines()` – file's contents as a list of lines

- (we will discuss lists in detail next week)

```
>>> f = open("hours.txt")
>>> f.readline()
'123 Susan 12.5 8.1 7.6 3.2\n'

>>> f = open("hours.txt")
>>> f.readlines()
['123 Susan 12.5 8.1 7.6 3.2\n',
'456 Brad 4.0 11.6 6.5 2.7 12\n',
'789 Jenn 8.0 8.0 8.0 8.0 7.5\n']
```

Line-based Input Template

- A file object can be the target of a `for ... in` loop
- A template for reading files in Python:

```
for line in open("filename"):
    statements
```

```
>>> for line in open("hours.txt"):
...     print(line.strip())   # strip() removes \n

123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

Writing Files

```
name = open("filename", "w")      # write
name = open("filename", "a")      # append
```

- opens file for <u>write</u> (deletes any previous contents) , or
- opens file for <u>append</u> (new data is placed after previous data)

`name.write(str)` – writes the given string to the file

`name.close()` – closes file once writing is done

```
>>> out = open("output.txt", "w")
>>> out.write("Hello, world!\n")
>>> out.write("How are you?")
>>> out.close()

>>> open("output.txt").read()
'Hello, world!\nHow are you?'
```

Exercise

- Write a function `remove_lowercase` that accepts two file names and copies the first file's contents into the second file, with any lines that start with lowercase letters removed.
  - example input file, `carroll.txt`:

```
Beware the Jabberwock, my son,
the jaws that bite, the claws that catch,
Beware the JubJub bird and shun
the frumious bandersnatch.
```

  - expected behavior:

```
>>> remove_lowercase("carroll.txt", "out.txt")
>>> print(open("out.txt").read())
Beware the Jabberwock, my son,
Beware the JubJub bird and shun
```

Exercise Solution

```
def remove_lowercase(infile, outfile):
    output = open(outfile, "w")
    for line in open(infile):
        if line[0].isupper():
            output.write(line)
    output.close()
```

lists

like Java’s arrays (but way cooler)

declaring:

- name = [value1, value2, ...] or
- name = [value] * length

accessing/modifying:

- name[index] = value

list indexing

lists can be indexed with positive or negative numbers (we’ve seen this before!)

index	0	1	2	3	4	5	6	7
value	9	14	12	19	16	18	24	15
index	-8	-7	-6	-5	-4	-3	-2	-1

list slicing

```
name[start:end]          # end is exclusive
name[start:]             # to end of list
name[:end]               # from start of list
name[start:end:step]     # every step'th value
```

- lists can be printed (or converted to string with str())
- len(list) returns a list’s length