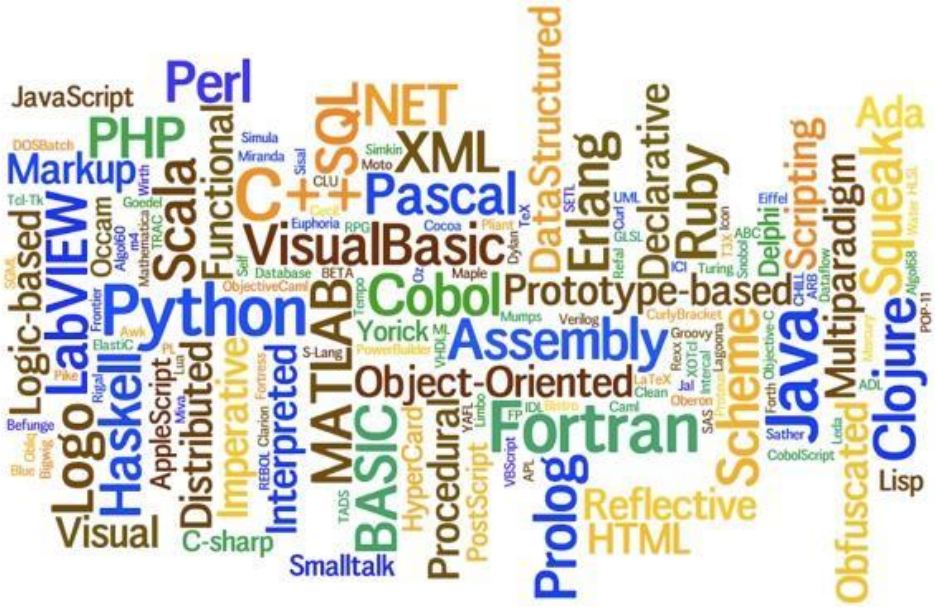


Programming Languages Python Syntax: Part I

Programming Languages

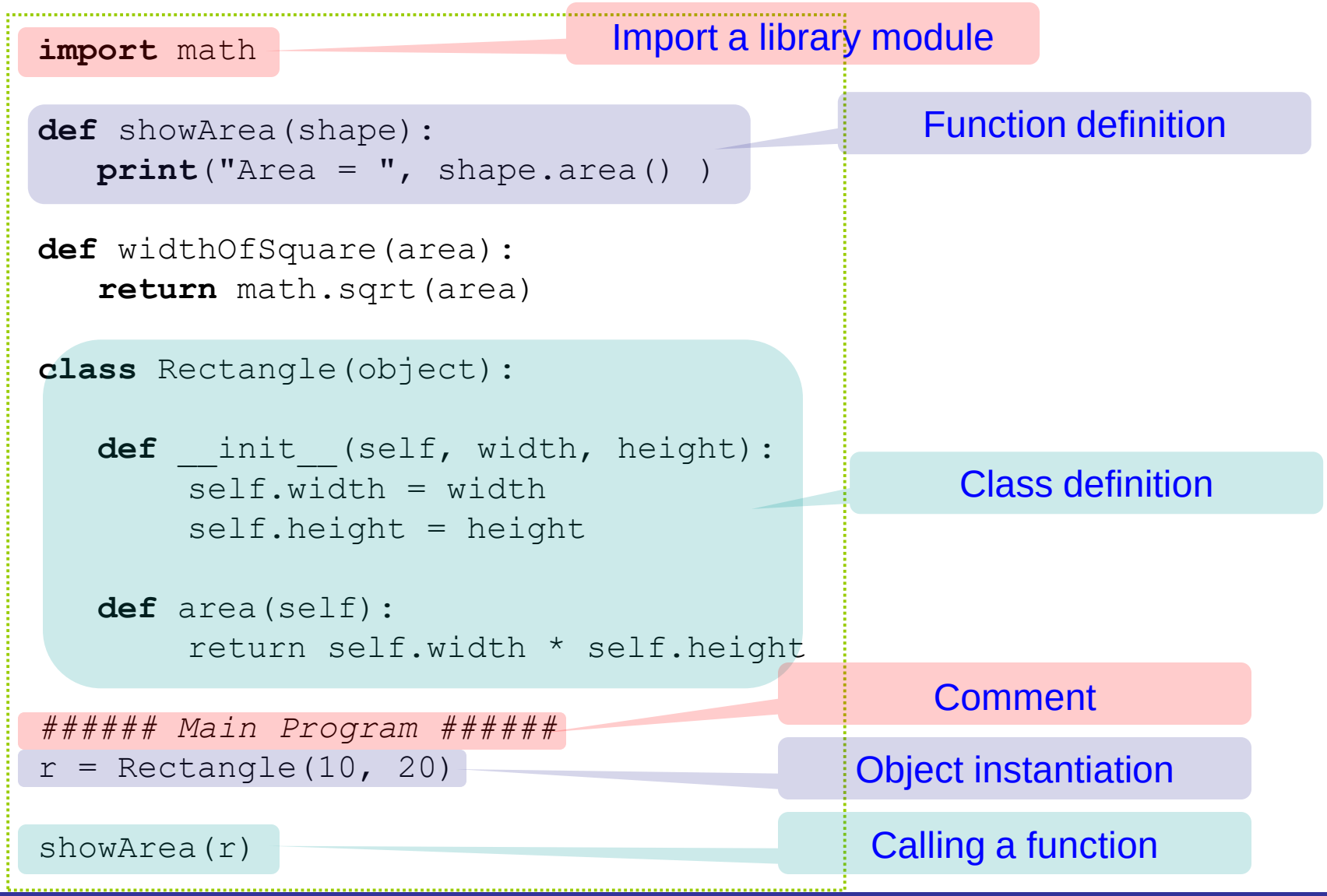
Dr. Tamer ABUHMED
College of Computing

SUNG KYUN KWAN UNIVERSITY

Outline

- Overview
- Built-in objects
- Data Types
- Arithmetic Operations
- Data Types Functions

Python At First Glance



Why use Python?

- Simple syntax: easy to learn *and* remember
- Portable
- Flexible
- Large standard library
- Short development time
- Lots of 3rd party tools/add-ons
- Many good implementations:

  CPython, PyPy, IronPython, Jython
- Active open-source community

Similarities to Java

- Everything inherits from "object"
  - Has numbers, functions, classes, …
  - Everything is first-class
- Large standard library
- Garbage collection
- Introspection, serialization, threads, net,…

Python vs. Java/C++/C

- Typing: strong, but dynamic
  - Names have no type
  - Objects have type
- No name declarations
- Sparse syntax
  - No { } for blocks, just indentation
  - No ( ) for if/while conditions
- Interactive interpreter
- # for comments (like Perl)

```
// this is Java
if (x < 10)
{
  x = x + tmp;
  y = y * x;
}
System.out.println(y);
```

Java

```
# this is Python
if x < 10:
  x = x + tmp
  y = y * x
print( y )
```

Python

Python 3: print("hello")

There are some differences between Python 2.x and Python 3 syntax.

- `print` is a function in Python 3, which uses parenthesis:

Python 3.x:

```
print("hello")
```

Python 2.x:

```
print "hello"
```

print

- `print` : Produces text output on the console.

- Syntax:

```
print "Message"
print Expression
```

  - Prints the given text message or expression value on the console, and moves the cursor down to the next line.

```
print Item1, Item2, ..., ItemN
```

  - Prints several messages and/or expressions on the same line.

- Examples:

```
print("Hello, world!")
age = 45
print("You have", 65 - age, "years until retirement")
```

Output:

```
Hello, world!
You have 20 years until retirement
```

Hello, World!

Java

```
class Hello
{
public static void Main(string args)
{
   System.out.println("Hello, World");
}
}
```

Python

```
print("Hello, World")
```

Variables

name *x* means 23

now it means 'foo'

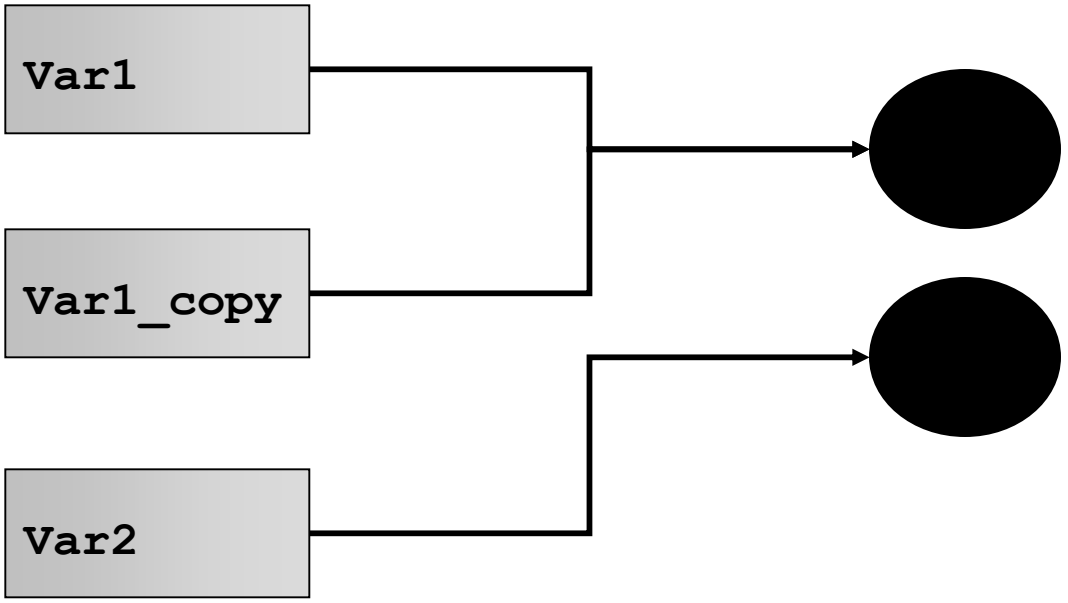
x is undefined

```
>>> x = 23
>>> print(x)
23
>>> x = 'foo'
>>> print(x)
foo
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>>
```

Variables

Variable is a reference to an object

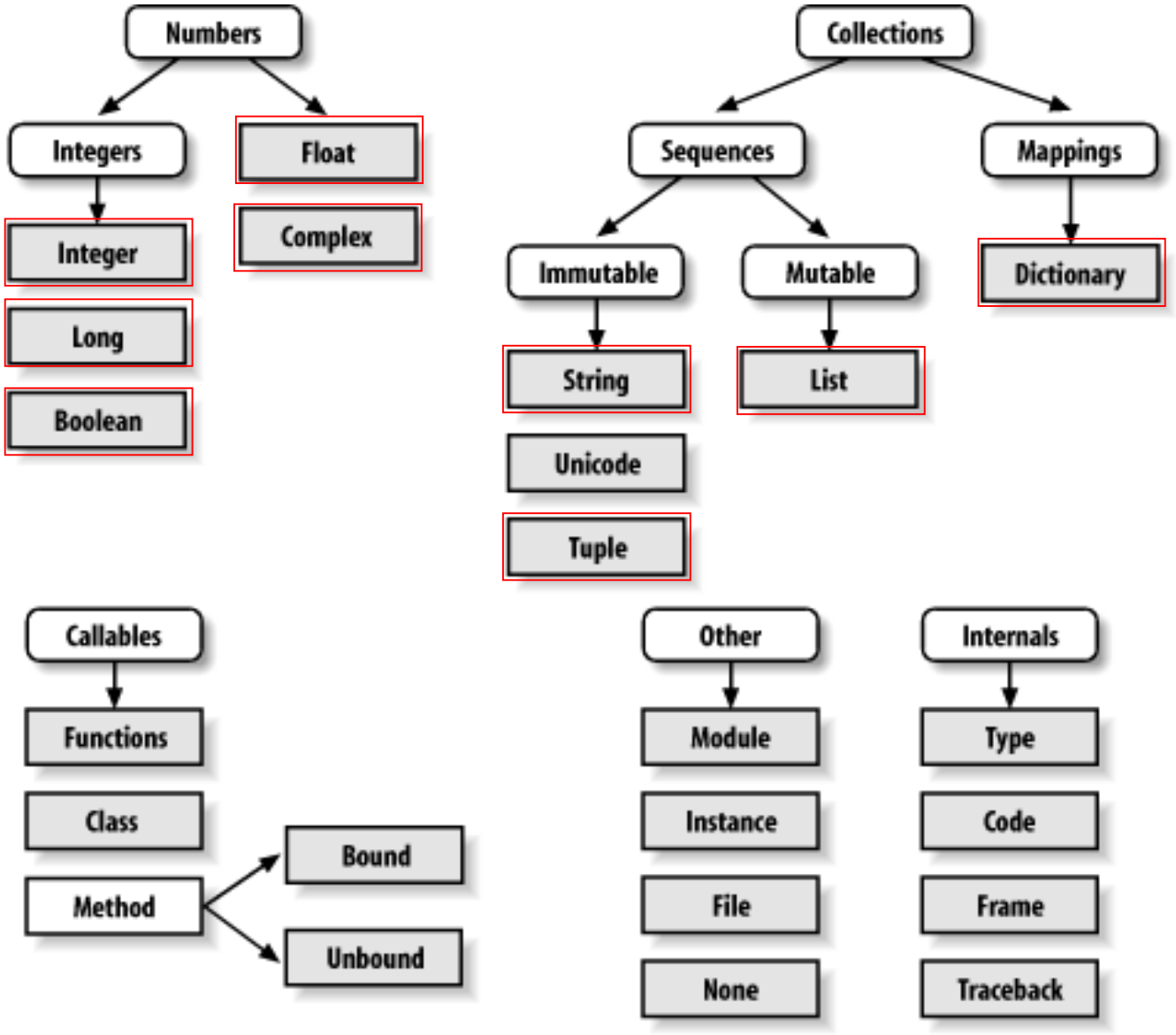
- not a value
- more than one variable can refer to the same object



Numeric Types

- Integers
  - Generally signed, 32-bit
- Long Integers
  - Unlimited size
  - Format: <*number*>L
  - Example: `4294967296L`
- Float
  - Platform dependant “double” precision
- Complex
  - Format: <real>+<imag>j
  - Example: `6+3j`

Python's built-in type hierarchy



Strings

A sequence of characters enclosed in quotes

3 ways to quote strings:

```
'Single Quotes'
"Double Quotes"
"""Triple Quotes""" or '''triple quotes'''
```

- Triple quotes can span multiple lines

Examples

```
>>> print('This string may contain a "')
This string may contain a "
>>> print("A ' is allowed")
A ' is allowed
>>> print("""Either " or ' are OK""")
Either " or ' are OK
```

`input` for reading input

`input( [`*`prompt`*`] )`

- Print *prompt* and return user's input as a string
- a built-in function

Example

```
>>> reply = input('Are you awake? ')
Are you awake? not sure
>>> print( reply )
not sure
```

Arithmetic Operations

operators: **+ - * / // ** % abs**

Examples:

```
>>> 5 + 3           # Addition
8
>>> 2 ** 8          # Exponentiation
256
>>> 13 // 4          # Integer (Truncating) Division*
3
>>> float(13) / 4 # Float Division
3.25
>>> 13 % 4          # Remainder
1
>>> abs(-3.5)       # Absolute Value
3.5
```

* 13/4 performs float division in Python 3.x

Boolean comparisons

Comparison: **< <= > >= == != <>**

Results in 1 (true) or 0 (false)

Example

```
>>> 4 > 1.5
true
>>> 'this' != 'that'
true
>>> 4+3j == 4-2j
false
>>> '5' == 5
 false
>>> 0 < 10 < 20
true
```

Boolean Operations

Operators: **`and`** **`or`** **`not`**

Standard Boolean Algebra

i_1	i_2	i_1 **and** i_2	i_1 **or** i_2
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

i_1	**not** i_1
1	0
0	1

Boolean values

`True`: any non-zero, non-null value.

`False`: `None` (null)

empty string

empty list

0

```
s = "hello"
if s:
   print("true")
lst = []
if lst:
   print("list is not empty")
```

Boolean Expressions

```
>>> 1 == 1 and 2 >= 3
False
>>> 1 == 1 or 2 >= 3
True
>>> not 5.3 != 2.2    # same as: not (5.3 != 2.2)
False
>>> 2 and '23' > '11' or 0
True
```

Math commands

- Python has useful commands for performing calculations.

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`, `***value2***`)`	larger of two values
`min(`***value1***`, `***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

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

- To use many of these commands, you must write the following at the top of your Python program:

```
from math import *
```

Building strings

Concatenation (+): `string1 + string2`

Example:

```
>>> 'Rockefeller' + 'University'
'RockefellerUniversity'
```

Repetition (*): `string * number`

Example:

```
>>> 'dog' * 5
'dogdogdogdogdog'
```

String Formatting

C-Style formatting (extended `printf`):

```
"format string" % (arg1, arg2, ...)
```

```
>>> "%i %s in the basket" % (2, "eggs")
'2 eggs in the basket'
>>> x = 2.0/9.0
>>> "%f to 2 dec places is %.2f" % (x, x)
'0.222222 to 2 decimal places is 0.22'


>>> length = 5
>>> obj = "fence"
>>> "Length of %(obj)s is %(length)i" % vars()
'Length of the fence is 5'
```

String Format Codes

Format codes begin with "%":

```
import math
x = 10
"x is %f" % x
"pi is %.8f" % math.pi
"pi is %12.6f" % math.pi
eps = 1.0E-17
"eps is %f (%g)" % (eps, eps)
```

String Formatting using `.format`

```
>>> "{0} {1}  in the basket".format(2, "eggs")
'2 eggs in the basket'
>>> x = 2.0/9.0
>>> "{0} to 2 dec places is {0:.2f}".format(x)
'0.222222 to 2 decimal places is 0.22'
"{0} to {1} dec places is {0:.{1}f}".format(x,3)
'0.222222 to 3 decimal places is 0.222'
>>> name = "James Bond"
>>> id = 7
>>> "{0:12s} is {1:03d}".format(name,id)
'James Bond   is 007'
```

Python format mini-language reference:

https://docs.python.org/3.4/library/string.html#format-specification-mini-language

String functions

`s = '''Now is the time for all good men'''`	Multi-line strings (triple quote)
`list = s.splitlines()`	return list of lines in string
`s.lower()`	to lowercase
`s.upper()`	to uppercase
`s.title()`	title case
`s.index('me')`	index of first occurrence, throw exception if substring not found
`s.count('me')`	count occurrences
`s[1:10]`	slice, just like list slice
`s.replace("men","people")`	replace substring.

String format functions

`>>> "Hello".ljust(8)` `"Hello   "`	Left justify to given length.
`>>> "Hello".rjust(8)` `"   Hello"`	Right justify.
`>>> "Hello".center(8)` `" Hello  "`	Center, of course.

`>>> u = "Bird"`
`>>> "Hello {0}".format(u)`
`'Hello Bird'`
	Format using template.

`type` determines type of Object

Determine the type of an object

Syntax: `type(object)`

Examples

```
>>> type(2.45)
<type 'float'>
>>> type('x')
<type 'str'>
>>> type(2**34)
<type 'long'>
>>> type(3+2j)
<type 'complex'>
```

Testing the type

```
if type(x) is int:
  print("x is an integer")
```

Type Conversions

Functions to convert between types:

`str()` `int()` `float()` `complex()` `bool()`

```
>>> str(0.5)
'0.5'
>>> float('-1.32e-3')
-0.00132
>>> int('0243')
243
>>> int(2**100)
1267650600228229401496703205376
>>> bool('hi')
True
>>> bool('False')     # any non-zero, non-null is true
True
```

Built-in Data Structures

- List l = [2, 3, 5, 8]
- Tuple (read-only list) t = (2, 3, 5, 8)
- Set s = { 2, 5, 3, 8 }
- Dictionary (key-value map) d = {"two":2, "three": 3, ...}