

# INTRO TO PYTHON

## SECTION 1



Emily Zhang  
Senior Software Engineer

# WELCOME

Girl Develop It is here to provide affordable and accessible programs to learn software through mentorship and hands-on instruction.

- **We are here for you**
- **Every question is important**
- **Help each other**
- **Have fun!**

# Outline

- ➔ **What is programming, Why Python**
- ➔ **Development Environment**
- ➔ **Variables and arithmetic**
- ➔ **In-Class Project**

# Part 1: Programming in Python

# WHAT IS PROGRAMMING?

Programming is like a cooking:

- Crack an egg into a bowl
- Whisk the egg
- Put a pan on medium heat
- Grease the pan
- Pour the eggs into the pan...

Computer Programming:

- Load some data
- Transform some data into result data
- Send the result data to the monitor for display
- Send the result data over the internet























# PYPL Popularity of Programming Language

Worldwide, Jan 2018 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	21.2 %	-1.4 %
2		Python	19.3 %	+4.9 %
3		PHP	8.0 %	-1.7 %
4	↑	Javascript	7.9 %	+0.1 %
5	↓	C#	7.5 %	-1.0 %
6	↑	C++	6.3 %	-0.7 %
7	↓	C	6.3 %	-0.8 %
8	↑	R	3.9 %	+0.6 %
9	↓	Objective-C	3.7 %	-0.7 %
10		Swift	3.1 %	+0.2 %

<http://pypl.github.io/PYPL.html>, Ranking based on Google search result

# Spectrum IEEE 2017 Top Programming Language

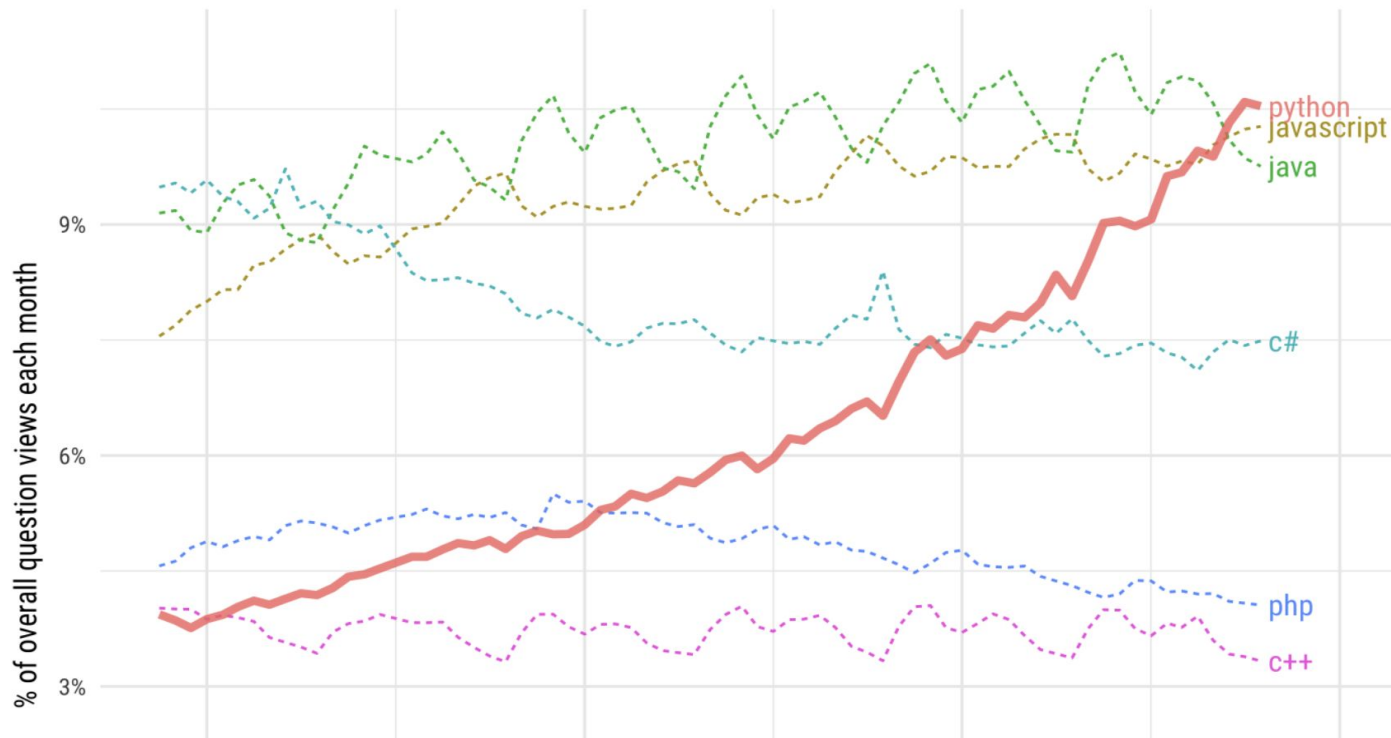
Language Rank	Types	Spectrum Ranking
1. Python	 	100.0
2. C	  	99.7
3. Java	  	99.5
4. C++	  	97.1
5. C#	  	87.7
6. R		87.7
7. JavaScript	 	85.6
8. PHP		81.2
9. Go	 	75.1
10. Swift	 	73.7

<https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages>

# StackOverflow Popular Language

## Growth of major programming languages

Based on Stack Overflow question views in World Bank high-income countries



<https://stackoverflow.blog/2017/09/06/incredible-growth-python/>



# What is Python Used For?

From <https://www.python.org/>

- **Web Development**

- Django, Pyramid, Bottle, Tornado, Flask, web2py

- **GUI development**

- tkInter, PyGObject, PyQt, PySide, Kivy, wxPython

- **Scientific & Numeric**

- Numpy, SciPy, Pandas, IPython

- **Software Development**

- Buildbot, Trac, Roundup

- **System Administration**

- Ansible, Salt, OpenStack

# Python: More Than Scripting Language

## ➤ Python: For Scripting

- batch file/text processing
- web crawling

## ➤ Python: Dynamic Typed Language

- Unlike C, Java, C++
- Clean syntax, easy to read

## ➤ One of the most desirable skills in job market

*All data below comes from [www.indeed.com](http://www.indeed.com) using search in United States, Feb 2018*

- Python: 46k job listings
- Java: 59k job listings
- C++: 31k job listings
- Data analyst: 78k job listings
- Data scientist: 24k job listings

# Python: Clean Syntax

## # Java

# hello world

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("hello world");  
    }  
}
```

# string operation

```
public static void main(String[] args) {  
    String test = "compare Java with Python";  
    for(String a : test.split(" "))  
        System.out.print(a);  
}
```

## # python

# hello world

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

# string operation

```
a="compare Python with Java"  
print(a.split())
```

More: <https://www.programcreek.com/2012/04/java-vs-python-why-python-can-be-more-productive/>

# WHO IS USING PYTHON?

- [Google](#)
- [Facebook](#)
- [Instagram](#)
- [Spotify](#)
- [Quora](#)
- [Netflix](#)
- [Dropbox](#)
- [Reddit](#)

# **Part 2: Development Environment**

# COMMAND LINE, PYTHON SHELL, TEXT EDITORS

	Description
<b>Text Editor</b>	<ul style="list-style-type: none"><li>• A program that opens text files and allows the user to edit and save them. (Different than a word processor). PyCharm, Sublime, Emacs, Vim, .....</li><li>• IDE: integrated development environment, or, advanced text editor, like <b>PyCharm Community Edition</b></li></ul>
<b>Terminal</b>	<p>A program that has a command line interface and issues commands to the operating system.</p> <ul style="list-style-type: none"><li>• mac: terminal, iterm</li><li>• windows: command prompt</li><li>• some IDE also provides terminal</li></ul>
<b>Python Shell</b>	<p>A command line program that runs inside of the terminal, takes Python code as input, interprets it, and prints out any results.</p>

# Development Environment - Practice 1

- ★ Install Python

- ★ Test Python Installed

\$ python3 --version      # on some computers, python --version

- ★ Terminal: Type 'python3' (or 'python') and hit enter.

\$ python3      # to enter Python Shell, on some computers, type 'python'

>>> exit()      # to get out of Python Shell

- ★ Terminal(Command Line) Navigation: Bash Commands (**advanced**)

- cd <path>      # change directory

- mkdir <dir\_name>      # create a new directory

- pwd      # show present working directory

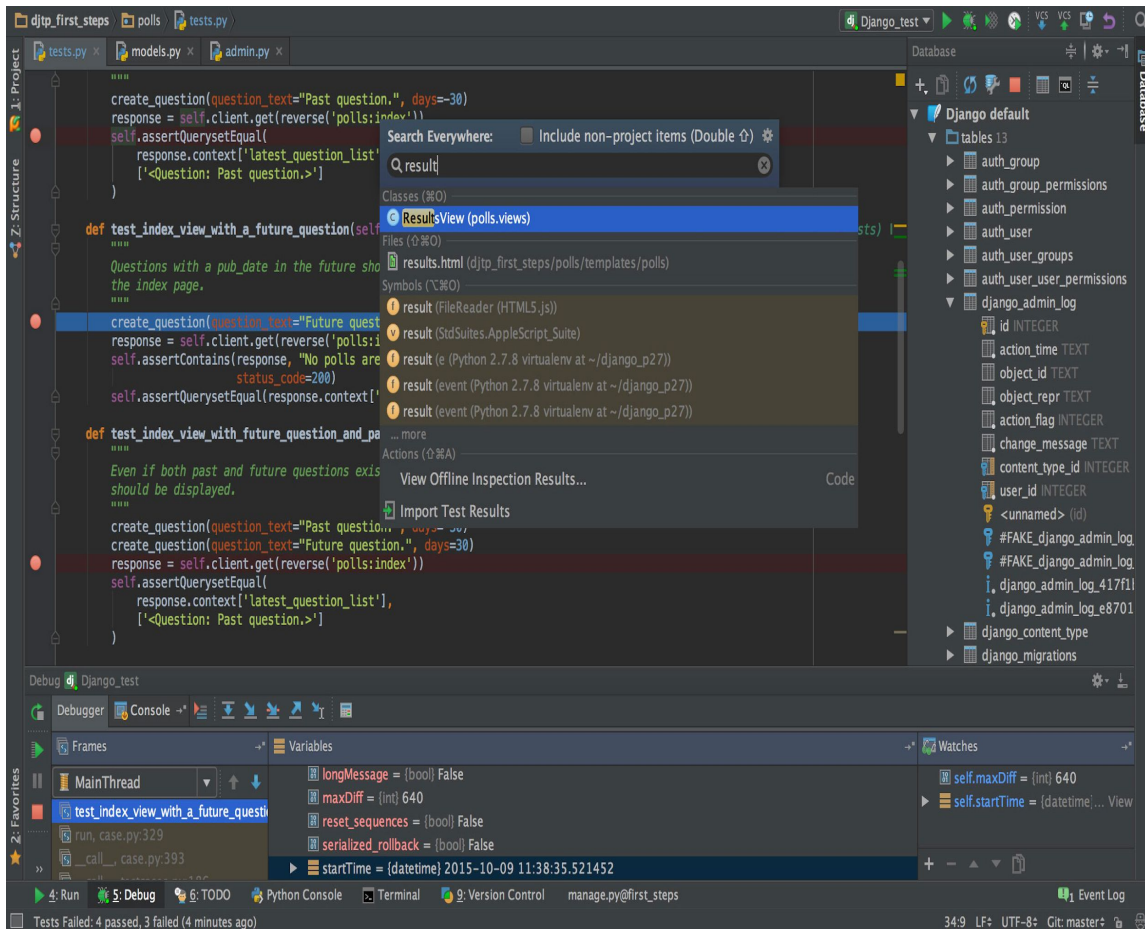
# Terminal | Bash -- Advanced

It's good to know some bash commands, they help us navigate on command line quickly. But not knowing them are not show stoppers, there are other ways to navigate or create directory too. All commands below are for Mac/Linux, some may not work for Windows.

- **pwd**  
show present working directory
- **ls**  
list current directory
- **cd <path>**  
change directory to <path>
- **mkdir**  
make a new directory
- **man <command>**  
get help on a command



# Working With PyCharm



## PyCharm

- easy navigation
- create project
- create file
- color highlighting
- edit code
- run code
- search code, find usage
- refactor code
- 'terminal' to run commands
- python 'console' to try code
- database integration
- version control integration
- ...more...

## Working With PyCharm - Practice 2

- **Terminal, make a directory 'gdi\_python', commands below are for Mac**

```
$ mkdir gdi_python      # mkdir: create a new directory
$ cd gdi_python         # cd: enter new directory
$ touch test1.py        # touch: create an empty python file called 'test1.py'
```

- **start a project on PyCharm called `gdi_python`**

- **use PyCharm to edit test.py**

```
print('hello world!')
print(2+3)
```

- **Run code on terminal, on some computers use 'python' not 'python3'**

```
$ python3 test1.py
```

- **Run code on PyCharm**

- ◆ right click, **'Run test1'**

# Advanced Learning

- **Bash**

- Google "[bash tutorial](#)", example: <https://astrobiomike.github.io/bash/basics>
- Learn to do more tasks with command line

- **PyCharm**

- Google: "[PyCharm Tutorial](#)" and get familiar with PyCharm
- Learn to config PyCharm, set PyCharm's color scheme to your desire

- **Python**

- Google '[example python code](#)' and copy some code to PyCharm and test run
  - note some python code might be written in Python2, not Python3
- PyCharm provides '[debugger](#)' tool, google '[pycharm debug tutorial](#)'

# Part 3: Numbers & Strings

## # Practice 3

```
>>> x = 15 # "=" sign states for 'assignment'
```

```
>>> y = 4 # the hash "#" is line comment
```

```
>>> x + y
```

```
>>> x - y
```

```
>>> x * y
```

```
# /: division, // : floor division
```

```
>>> a = x / y; print(a); type(a)
```

```
>>> b = x // y; print(b); type(b)
```

```
# %: modulus, remainder
```

```
>>> x % y # 15%4 = 3, 9%2=1
```

```
# x to the y th, x^y
```

```
>>> x ** y
```

```
# get type of variables
```

```
>>> type(x) # <class 'int'>
```

```
>>> type(y)
```

# Practice 1 - Numbers

# Arithmetic operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Arithmetic operators in Python

Operator	Meaning	Example
+	Add two operands or unary plus	$x + y$ +2
-	Subtract right operand from the left or unary minus	$x - y$ -2
*	Multiply two operands	$x * y$
/	Divide left operand by the right one (always results into float)	$x / y$
%	Modulus - remainder of the division of left operand by the right	$x \% y$ (remainder of $x/y$ )
//	Floor division - division that results into whole number adjusted to the left in the number line	$x // y$
**	Exponent - left operand raised to the power of right	$x ** y$ (x to the power y)

# Number Data Types

- **int**
  - any length
  - Be aware of return data type  
`>>> 4 / 2` # returns float 2.0
- **float**
  - Example: 3.14
  - accurate up to 15 decimal places (the 16th place is inaccurate)
- Complex, Fractions, Decimal
  - Python offers more number data types, won't cover in this course.
  - **Decimal:**
    - More precise than float
    - Less performant than float

# Practice 4

```
>>> a = int()
>>> a                # 0
>>> print(a)         # 0
```

```
>>> a = float()
>>> a                # 0.0
```

```
>>> a = 16 ** 24
>>> type(a)          # <class 'int'>
```

# auto type conversion

```
>>> a = 3 + 2.0      # mix int/float -> float
>>> type(a)          # <class 'float'>
```

# type conversion, type casting

```
>>> int(3.0)         # 3
>>> float(7)         # 7.0
```

# built-in functions 'round'

```
>>> round(3.4)        # 3
>>> round(3.718, 2)   # 3.72
```



# Numbers Summary

## ➤ Python Number Data Types

- int, float

## ➤ Arithmetic Operators

- +, -, \*, /, //, \*\*, %

## ➤ Python Builtin Functions

int, float, type, print, round

## ➤ Get help in Python

- `>>> print(round.__doc__)`
- `>>> help(round)` # press 'q' to exit help mode
- **Google!**

## 2. Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
<code>abs()</code>	<code>dict()</code>	<code>help()</code> ★	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code> ★	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code> ★	<code>iter()</code>	<code>print()</code> ★	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code> ★
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code> ★	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

# Practice 5 -- Strings

- **String Length & Concatenation**

```
>>> hello = 'hello'      # String literals can use single quotes
>>> world = "world"      # or double quotes; it does not matter.

>>> hello                 # Prints "hello"
>>> len(hello)            # String length; prints "5", built in function 'len'

>>> hw = hello + ' ' + world  # String concatenation, form a new string
>>> hw                     # "hello world"
```

- **String Index & Slicing**

```
>>> world[2]              # the 2nd element in the string, note we count from 0, returns 'r'
>>> world[:2]             # string slicing, [:2] is same as [0:2], returns "wo", a new string
>>> world[0:2]
```

## String Operation: +, \*, index, slicing, len -- Practice 6

```
>>> 'home' + 'work'      # '+': concatenation
homework
```

```
>>> 'work' * 2           # '*': repetition
workwork
```

```
>>> grades = 'ABCDE'    # indexing, python index starts with 0 to length-1
>>> grades[0]
'A'
```

```
>>> grades[2:4]          # 'CD', slicing
>>> grades[:3]           # 'ABC'
>>> grades[2:]           # 'CDE'
```

```
>>> for i in [0, 1, 2, 3, 4]:      # advanced material
...     print("i=", i, ", grades[i]=", grades[i])  # 4 space indent before 'print'
```

# Strings - Practice 7

- **String Formatting -- Old Style Formatting**

```
>>> myname = 'Emily'
>>> new_var = 'Hello %s' % myname      # string formatting, there are other ways to do it too
>>> new_var                           # 'Hello Emily'
```

```
>>> myname = 'John'
>>> new_var = 'Hello %s' % myname
>>> new_var                           # 'Hello John'
```

"%s" sign is called a "**placeholder**", that we'll add value later, usually from a variable

```
>>> for name in ('Emily', 'John', 'Mary'):      # we'll cover 'for loop' in next lecture
...     print("Welcome %s to GDI!" % name)      # note, 4 spaces before 'print' for indentation
```

# String Operators: Summary

+	Concatenation - Adds values on either side of the operator	'Hello' + ' Python' → 'Hello Python'
*	Repetition - Creates new strings, concatenating multiple copies of the same string	'Hello' * 2 → 'HelloHello'
[n]	Index - get character at n-th index Index starts from 0  Index for 'Hello' : 0('H'), 1('e'), 2('l'), 3('l'), 4('o')	"Hello"[1] → 'e'
[n:m]	Range Slice - Gives the characters from the given range 'Hello'[2:] → 'llo'      'Hello'[:4] → 'Hell'	"Hello"[2:4] → 'll'

# Quotes

- Single quote & Double quote
  - Interchangeable
  - Many people prefer single quotes, it “LOOKS” neater.
- Triple quotes (single or double)
  - For **multiple line strings**

```
>>> multi_line_variable = """
```

```
... The Zen of Python, by Tim Peters
```

```
...
```

```
... Beautiful is better than ugly.
```

```
... Explicit is better than implicit.
```

```
... Simple is better than complex.
```

```
... Complex is better than complicated.
```

```
... Flat is better than nested.
```

```
... Sparse is better than dense.
```

```
... Readability counts.
```

```
... """
```

```
...
```

```
>>> import this    # will yield the beautiful “Python Zen” above
```

# Strings Summary

## → String Operations

- ◆ + for concatenation
- ◆ \* for repetition
- ◆ [n] get character at n-th index
- ◆ [n:m] slice string into a new string, with length m-n

## → Briefly Introduced

- ◆ String Formatting using %: "hello %s" % "world" → "hello world"
- ◆ Quotes: single, double, triple single, triple double



# Numbers & Strings: Advanced Learning

- **Numbers & Strings:**  
[https://www.learnpython.org/en/Variables\\_and\\_Types](https://www.learnpython.org/en/Variables_and_Types)  
[https://www.learnpython.org/en/Basic\\_Operators](https://www.learnpython.org/en/Basic_Operators)
- **More Strings:**  
[https://www.learnpython.org/en/Basic\\_String\\_Operations](https://www.learnpython.org/en/Basic_String_Operations)

# Part 4: Project

# input() & eval()

## → eval()

- ◆ built in function eval(), will take input strings, outputs int or float, **string -> number**
- ◆ `>>> eval("5")` # returns integer 5
- ◆ `>>> eval("5.0")` # returns float 5.0

## → input()

- ◆ built in function input()
- ◆ `a = input()` will prompt user to input something.
- ◆ `>>> a = input('please input an integer: ')`
- ◆ **Question:** what data type is "a" above, after user inputs something?

# jupyter notebook - optional

## → Jupyter notebook

a simple and interactive tool to write python code. google 'jupyter notebook tutorial'

## → Install: Mac: pip3 install jupyter

\$ jupyter notebook                      # will open on browser a new tab

## → online

<http://jupyter.org/try>, click on "Try Jupyter with Python"

# PROJECT -- change\_counter.py

1. create python file `change_counter.py` from terminal or in PyCharm
2. use '`input()`' function to prompt user to input how many quarters, example on python shell:  

```
>>> quarters = input('Please input how many quarters: ')
```
3. Write code in a text editor, prompt users to input number of: quarters, dimes, nickels, pennies, and save the four numbers in four variables
4. Run code from terminal: `python3 change_counter.py`
5. Calculate the total amount we have in cents, using arithmetic operators like: `+`, `*`
6. Print to screen: "Total amount of money is: xxx cents", please use old style string formatting  

```
>>> print('Total amount of money is: %s cents' % total)
```
7. Print to screen: "Total amount of money is: x.xx dollars", keep 2 decimal places, using old style formatting

# PROJECT -- change\_counter.py -- Solution

```
# 'input' built-in function will return string type of number, for example '100',  
# to get the 'int' type, use 'eval' built-in function, that converts string -> number (int or float)  
quarters = input('Please input how many quarters: ')  
quarters = eval(quarters)
```

```
# having 2 functions (eval, input) on same line, is equivalent as in separate lines above  
dimes = eval(input('Please input how many dimes: '))  
nickels = eval(input('Please input how many nickels: '))  
pennies = eval(input('Please input how many pennies: '))
```

```
total = quarters * 25 + dimes * 10 + nickels * 5 + pennies * 1
```

```
# old style python string formatting  
print('Total amount of money is: %s cents' % total)
```

```
# new style python string formatting  
print('Total amount of money is: {} dollars'.format(total/100))
```

# Summary of Today

## → Development Environment

- ◆ terminal, command line, learn some Bash commands
- ◆ google 'PyCharm tutorial' and get familiar with PyCharm

## → Numbers

- ◆ Variables, assign value to a variable
- ◆ Type of Numbers: int, float
- ◆ Arithmetic Operation: +, -, \*, /, //, \*\*, %

## → Strings

- ◆ + for concatenation
- ◆ \* for repetition
- ◆ s[n] for indexing
- ◆ s[n:m] for slicing
- ◆ old style formatting: "Welcome %s!" % name