INTRO TO PYTHON

SECTION 2

Emily Zhang
Senior Software Engineer

Review of Lecture 1

- Arithmetic
- Variables
- Data types
- Text editor, command line, and python shell
- Homework

String Review - Practice 1

- **How to format the line below?**

  ```
  >>> a = 'hello'
  >>> b = 'world'
  >>> ???                    # use old / new style format, so it shows
  'This is hello and world!'
  ```

- **Formatting: Old Style**

  ```
  >>> ???
  ```

- **Formatting: New Style**

  ```
  >>> ???
  ```

String Review - Practice 1

- **How to format the line below?**

  -
    ```
    >>> a = 'hello'
    >>> b = 'world'
    >>> ???                    # use old / new style format, so it shows
    'This is hello and world!'
    ```
- **Formatting: Old Style**

  -
    ```
    >>> 'this is %s and %s!' % (a, b)
     'this is hello and world!'
    ```
- **Formatting: New Style**
  - `>>> 'this is {} and {}!'.format(a, b)`
  - `'this is hello and world!'`
  - `>>> 'this is {0} and {1}!'.format(a, b)`
  - `'this is hello and world!'`

Lecture 2 Outline

- ➔ **Boolean Expressions**
- ➔ **If Statement**
- ➔ **Loops**
  - ◆ For Loop
  - ◆ While Loop
- ➔ **Functions**

Boolean

Operators On Numbers: Practice 2

```
>>> 3 * 4 > 10                                    # returns: True
>>> 5 + 5 >= 10                                   # True
>>> a = (3 * 4 > 10) and (5 + 5 >= 10)

>>> print(type(a), '  ,  ', a)                    # print multi values, use ','
<class 'bool'>   ,   True

>>> b = (3 * 4 < 10) or (5 + 5 > 10) or (4 * 4 > 15) or (5 + 4 > 10)
>>> print(type(b), b)
<class 'bool'> True

>>> c = not (3 * 4 > 10)
>>> print(type(c));  print(c)       # multi statements on same line with ";"
<class 'bool'> False
```

Boolean Data Type

Python Basic Data Types

- Numbers : int, float, decimal, fraction, complex...
- String

- **Boolean Type**
  a. "**if statement**" uses "**boolean expression**"
  b. **Boolean**: Named after George Boole
  c. Two Boolean Values: **True / False** (or 1/0 in some other languages)

- In Later Lectures:
  a. List , Tuple, Set, Dictionary

Comparison 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`

Logical / Boolean Operators

Operator	Example	Result
and	(9 != 6) and (2 < 3)	True
or	(2 == 3) or (-1 < 5)	True
not	not (7 > 0)	False

Practice 3 - Boolean

# Boolean Operators

```
>>> x = True        # note the camel case: True, not true
>>> y = False

>>> x and y         #  False
>>> x or y          #  True
>>> not x           #  False
>>> not y           #  True
```

# Summary

True and True → True
True and False → False

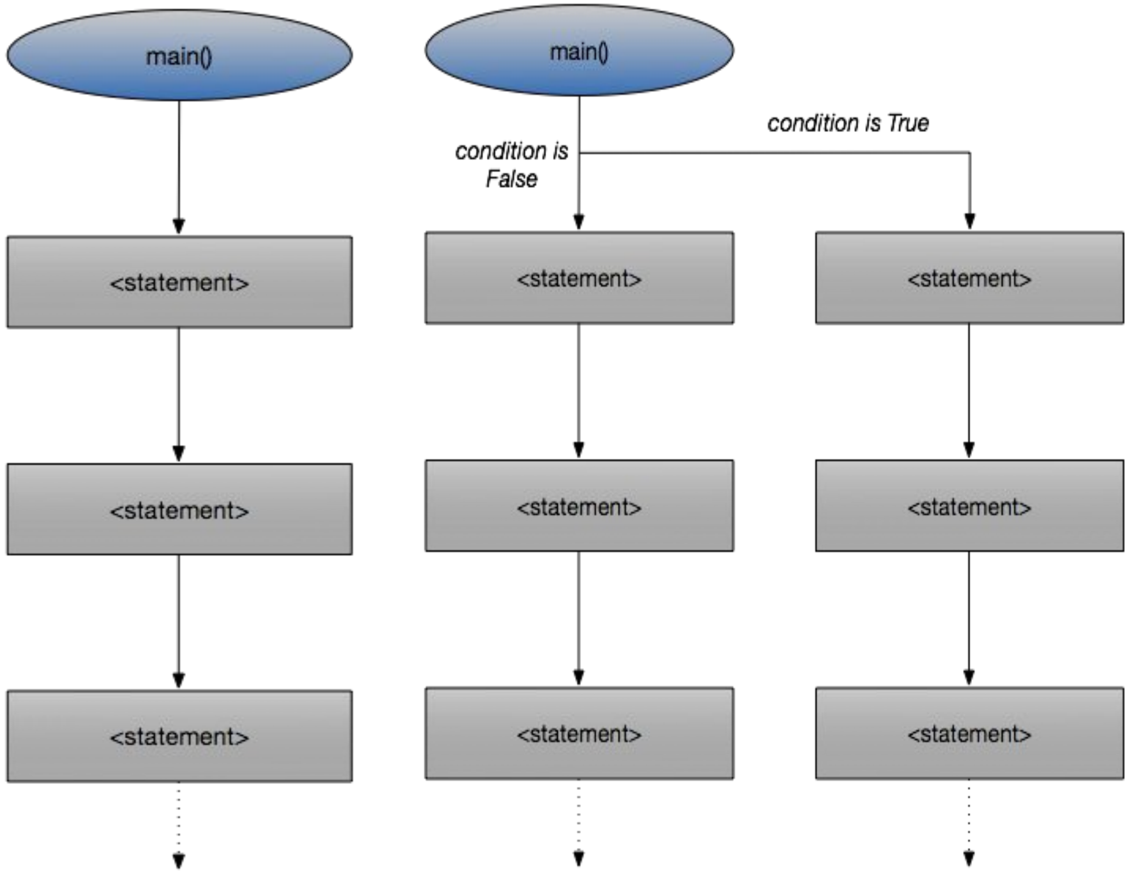
True or False → True

not True → False
not False → True

If Statement

Control Flow

- **if statement**
  - if
  - if / else
  - if / elif / else
  - nested if
- **if loop**
- **while loop**



if statement

# Format

```
# if
if <bool expression>:
    statement(s)


# if else
if <bool expression>:
    statement(s)
else:
    statement(s)
```

# Examples

- # if

```
>>> user_input = eval(input("Enter a positive number: "))
>>> if user_input < 0:
...       print("Your input %s is negative, exiting." % user_input)
```

- # if else

```
>>> my_var = 11
>>> if my_var > 10:
...     print("Value of a is greater than 10")              # indented
...  else :
...     print("Value of a is smaller than or equal to 10")  # indented
```

- **Colon**: **if / elif / else** statements have colon "**:**" in the end
- **Indentation**: The lines within **if / elif / else** are indented, common indentation is **4 spaces**

if statement: Execution

- **Code Example**:

```
if temperature > 90:                                # line 1
    print('too hot')                                # line 2
else:                                               # line 3
    print('not that hot')                           # line 4
print('Finished evaluating temperature')            # line 5
```

- **When <bool expression> is True**
  - **Executes**:   line: 1 → 2 → 5

- **When <bool expression> is False:**
  - **Executes**:   line: 1 → 3 → 4 → 5

if, elif, else

```
# format

 if <expression1>:
    statement(s)      # indented
 elif <expression2>:
    statement(s)      # indented
 elif <expression3>:
    statement(s)      # indented
 else:
    statement(s)      # indented
```

```
# example

var = 100
if var == 200:
   print('var == 200')
elif var == 150:
   print('var == 150')
elif var == 100:
   print('var == 100')
else:
   print('var=', var)

print("Good bye!")
```

Tutorial: https://docs.python.org/3/tutorial/controlflow.html
https://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ifstatements.html

if , elif , else

```
# if_else.py
var = 100                                  # executed line 1
if var == 200:                             # executed line 2
   print("1 - Got a true expression value")   # 'print' to debug
   print("var == 200\n")
elif var == 150:                           # executed line 3
   print("2 - Got a true expression value")
   print("var == 150\n")
elif var == 100:                           # executed line 4
   print("3 - Got a true expression value")   # executed line 5
   print("var == 100\n")                      # executed line 6
else:
   print("4 - Got a false expression value")
   print("var=%s\n" % var)

print("Good bye!\n")     # always execute since it's out of 'if'
```

Nested If -- advanced

```
# if_nested.py

    var = 100
    if var < 200:
       print("Expression value is less than 200")
       if var == 150:
          print("Which is 150")
       elif var == 100:
          print("Which is 100")
       elif var == 50:
          print("Which is 50")
       elif var < 50:
          print("Expression value is less than 50")
    else:
       print("Could not find true expression")

    print("Good bye!")
```

Practice 4:   Let's develop it !

Write a program that uses **if statements** to determine what to do given user input, the code below is an example. Save this file as **life_choice.py,** and execute this python file from terminal or IDE

```
health = 100
print("A vicious warg is chasing you.")
print("Options:")
print("1 - Hide in the cave.")
print("2 - Climb a tree.")
input_value = input("Enter your choice:")
if input_value == '1':                                          # note the colon
    print('You hide in a cave.')                                # indented 4 spaces
    print('The warg finds you and injures your leg with its claws')
    health = health - 10                                        # assignment from right to left
elif input_value == '2':
    print('You climb a tree.')
    print('The warg eventually looses interest and wanders off')
print("Game under construction. Come back later")
```

Loops

For Loops

```
>>> stocks = ['AAPL', 'FB', 'SNAP']              # list elements separated by “,”
>>> for s in stocks:                             # ‘for’ loop is the most common loop in Python
…       print('The next stock is: %s' % s)       # Prints each stock symbol on its own line.
…
The next stock is: AAPL
The next stock is: FB
The next stock is: SNAP
```

```
# form a new list by using a loop
>>> nums = [0, 1, 2, 3, 4]
>>> squares = []                        # forms an empty new list
>>> for n in nums:
...       squares.append(n ** 2)        # append new element to existing list
>>> squares                             # [0, 1, 4, 9, 16]
```

For Loops Explained

```
>>> stocks = ['AAPL', 'FB', 'SNAP']              # list elements separated by “,”
>>> for s in stocks:                             # ‘for’ loop is the most common loop in Python
…       print('The next stock is: %s' % s)       # Prints each stock symbol on its own line.
…
The next stock is: AAPL
The next stock is: FB
The next stock is: SNAP
```

- ❏ The variable ‘**s**’ above: just a variable inside of the ‘for’ block. It can be named anything.
- ❏ For loop has colon ":" similar to "if statement", and followed by indented block of code.
- ❏ for … in … , the entity after 'in' operator must be iterable: list, tuple, range, … It can not be a single value.

Practice 5: For Loops & If Statement

```
# Example
>>> nums = [0, 1, 2, 3, 4]
>>> squares = []                         # forms an empty new list using square brackets
>>> for n in nums:
...         squares.append(n ** 2)        # append new element to existing list
>>> squares                              # [0, 1, 4, 9, 16]
```

```
# Practice
>>> nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> even_numbers = []                    # forms an empty new list
>>> <write your lines here using if and for loop>     # hint: % for remainder

>>> even_numbers                         # [0, 2, 4, 6, 8, 10]
```

While Loop

```
# while_loop.py

input_value = input('Enter a positive integer:')

n = int(input_value)                    # use "eval" or "int" to convert string to integer

result = 1

while n > 1:

   result = result * n                  # assign right to left

   n = n - 1                            # whole 'while block' indented

print("The factorial of {0} is: {1}".format(input_value, result))        # 'format'
```

Loop Control: break, continue

- **continue:** skip the rest of the current iteration, jump to next iteration

```
# loop_continue_break.py
for letter in 'Python':     # 'string' is iterable, it will return one character back at a time
 if letter == 'h':
   continue
 print('Current Letter :', letter)              # will print P y t o n, one letter on a line
```

- **break:** jump out of the 'for' or 'while' loop

```
for letter in 'Python':     # First Example
 if letter == 'h':
   break
 print('Current Letter :', letter)              # will print P y t, one letter on a line
```

Tutorial: http://www.tutorialspoint.com/python/python_loop_control.htm

Functions

Functions

# function_example.py

```
# function with no input argument
def happy_birthday_1():
    name = 'Emily'
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear %s..." % name)
    print("Happy Birthday to you!")

# function with 1 input argument
def happy_birthday_2(name):
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear %s..." % name)
    print("Happy Birthday to you!")

happy_birthday_1()
print()
happy_birthday_2('Emily')
```

# execution result

```
Happy Birthday to you!
Happy Birthday to you!
Happy Birthday, dear Emily...
Happy Birthday to you!

Happy Birthday to you!
Happy Birthday to you!
Happy Birthday, dear Emily...
Happy Birthday to you!
```

Function Tutorial
https://www.learnpython.org/en/Functions

Function Explained

```
# function_example.py

# function with no input argument
def happy_birthday_1():
    name = 'Emily'
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear %s..." % name)
    print("Happy Birthday to you!")

# function with 1 input argument
def happy_birthday_2(name):
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear %s..." % name)
    print("Happy Birthday to you!")

happy_birthday_1()
print()
happy_birthday_2('Emily')
```

- **Define a Function**
  - def
  - function name: lower_case usually
  - (arg1, arg2, arg3, ...):
  - content of function: indented
- **Call a Function**
  - function_name(), no indent
  - if function definition has input argument, like 'name' in the example, pass in the argument
- **Practice:**
  - define a function that takes no input argument, and prints 1 sentence of your choice. Then call this function

Return from Function

```
# function_return.py
def get_product1(n, m):
    product = n * m

def get_product2(n, m):
    product = n * m
    return product

result = get_product1(5, 8)
print('result1=', result)
result = get_product2(5, 8)
print('result2=', result)
```

- **Local Variable | Scope | Namespace**

  Outside of function definition, variable 'product' no longer exist, So they're also called 'local variables'. In order to preserve the value of 'product', we need 'return' statement

- **'return'** statement

  pass value(s) back from the function to the caller

- some function don't have **'return'** statement, that's equivalent to 'return None'

Return from Function - More

```
def func1(n, m):
    n = 5 * m
    return
```

```
def func2(n, m):
    n = 5 * m
    return None
```

```
def func3(n, m):
    n = 5 * m
```

- ➔ Three functions above are identical in functionality, **all return 'None'.**

- ➔ **function naming**: similar to variable naming, prefers: lower_case_with_underscore

- ➔ Inside of one file, try to avoid having functions with same names

Functions

- Built-in functions
  - No need to import, just use them
  - int, float, str, list, input, eval, range
  - type, len, round, sum, max, len

- **User Defined Function**
  - def func_name(arg1, arg2):
  - indented function body

- Lambda function (advanced)
  - Also called 'Anonymous functions' or 'in line function'

    ```
    >>> f = lambda x: x**2
    >>> f(5)                              #  returns 25
    ```

- Call a function
  - function_name(), with arguments passing in, or no arg

Practice 6: Reverse a Sentence

- ➔ Objective
  - ◆ review: function, string, loop, input, print, len
- ➔ Write a function: reverse_sentence.py
- ➔ Prompt user to input a sentence, then reverse print to screen.
- ➔ Input: Python is a beautiful language!

  Output: !egaugnal lufituaeb a si nohtyP
- ➔ Hint
  - ◆ use 'input' function to ask for a sentence
  - ◆ get length of the input sentence
  - ◆ last last character index from the length of the sentence
  - ◆ have a counter variable to walk from last character, to the left
  - ◆ form a new sentence by string concatenation, in the end, print it

Function Execution

```
# function_execution.py

print('Line 3')

def func1():
   print('Begin func1')


def func2():
   print('Begin func2')
   return

# continue on the right
```

```
# function_execution.py

# continued from the left

def func3(x):
   print('Begin func3, input is: ', x)
   func1()
   func2()
   print('End func3')


print('Before calling func3...')
func3(10)
print('After calling func3...')
```

4 Steps Execution

1. The **calling program suspends execution** at the point of the call

2. The **parameters** in the definition of the function get assigned to the actual values from the caller

3. The body of the **function starts to execute**

4. After function finished executing, **return to the caller**, and continue executing to the next line

Intro: Debug

- **Print**
  - Especially useful when function is complex
  - print('this is line 123'), print('variable x=', x), print('inside of if statement')...

- **Debugger: ipdb, PyCharm Debugger, ...**
  - https://wiki.python.org/moin/PythonDebuggingTools
  - Especially useful for complex or recursive function
  - **install ipdb**: pip3 install ipdb
  - **PyCharm** comes with debugger, google **'PyCharm Debugger Tutorial'**

Function Types

- Function that <u>returns value(s)</u>

- Functions that <u>modifies input data in-place</u>
  - list/dict are mutable, input list/dict argument can be modified 'in-place'
  - function may seem like 'return None' but actually input list/dict is changed

- Function that has <u>'side effect'</u>
  - print to screen, sending an email, write to file, cleanup disk space

- Mix of above types

Function Summary

- ★ Don't Repeat Yourself! **(DRY Principle)**
- ★ help structure or modularize programs
  - can be called multi times with different input
- ★ function execution:
  - runs caller, function parameter passed, function runs, back to caller
- ★ scope of variable
  - variables inside of function are **local variables**, invalid out of this func

DRY: https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

Thank You