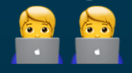
LAB 01

Functions, Control

Mingxiao Wei
mingxiaowei@berkeley.edu

Aug 29, 2022

LOGISTICS 

- Lab 00 due Wed 08/31
  - Make sure to finish lab 00 before doing any other assignment!
- Lab 01 also due Wed 08/31
- Homework 01 due Thu 09/01
  - In general, it's recommended to finish lab first, then do the homework
- Look for study buddy or project partners? Talk to your neighbors or check out Ed post #128!
  - The first project is coming soon! You can choose to work alone or in group of 2 

USING PYTHON 🐍 🤦

❗ Make sure to navigate to the correct directory ans save your work ❗

- `python3 <filename>`
  - run the code in the file you provide and return you to the command line
- `python3 -i <filename>`
  - runs the code in the file, then opens an interactive session where you can run Python code line by line and get immediate feedback.
  - To exit, type `exit()` into the interpreter prompt. You can also use the keyboard shortcut `Ctrl-D` on Linux/Mac machines or `Ctrl-Z` Enter on Windows
- `python3 -m doctest <filename>`
  - Runs doctests in a particular file. Doctests are surrounded by triple quotes (`"""`) within functions.
  - Each test in the file consists of `>>>` followed by some Python code and the expected output (though the `>>>` are not seen in the output of the doctest command).
- `python3`
  - opens an interactive session without referring to any source file

USING OK 👌

❗ Make sure to navigate to the correct directory ans save your work ❗

- Most of the time, you don't have to worry about anything other than copying and pasting the ok commands provided in the spec :D
- `python3 ok -q <specified function>`
  - run tests for a specified function
- `python3 ok`
  - run all tests; only failed ones will not be shown
- `python3 ok -v`
  - run all tests and show all results, including the ones that passed
- `python3 ok --score`
  - run all tests and show your sore
- `python3 ok --submit`
  - REMEMBER TO RUN THIS EVERY TIME YOU FINISH AN ASSINGMENT!
- `print("DEBUG:", <content>)`
  - Print content for debugging purposes - printed lines that start with `DEBUG:` will be ignored by the ok autograder

KEYBOARD SHORTCUTS 📍

Nothing here is necessary for 61A, but they make your life easier!

- In the terminal, use ⬆️ or ⬇️ to retrieve the previous/next command you've entered - useful when repeatedly running tests
- Select lines of code and:
  - `cmd/ctrl + ?` to comment/uncomment the selected lines
  - `tab` to move them one indentation level inward
  - `shift + tab` to move them one indentation level outward
- `cmd/ctrl + s` save
- `cmd/ctrl + z` undo
- `cmd/ctrl + a` select all
- `cmd/ctrl + x` cut
- `cmd/ctrl + c` copy
- `cmd/ctrl + v` paste

DIVISION, FLOOR DIV, AND MODULO

True Division: `/` (always returns a decimal number)	Floor Division: `//` (rounds down to the nearest integer)	Modulo: `%` (remainder)

`>>> 3 / 4`
`0.75`
 	
`>>> 6 / 3`
`2.0`
 	
`>>> 1 / 0`
`ZeroDivisionError`
	`>>> 3 // 4`
	`0`
	
	`>>> 6 // 3`
	`2`
	
	`>>> 1 // 0`
	`ZeroDivisionError`
		`>>> 3 % 4`
		`3`
		
		`>>> 6 % 3`
		`0`
		
		`>>> 1 % 0`
		`ZeroDivisionError`

`x % y == 0` evaluates to `True` when `x` is divisible by `y`

⇒ useful when checking for divisibility or even numbers

FUNCTIONS 

FUNCTIONS

- Abtraction of some executions
- A function takes some arguments (or no argument), and returns some value, or `None` if there's no return statement.
- To apply a function to some arguments, we use call expressions

CALL EXPRESSIONS

Anatomy: `<operator>(<operand1>, <operand2>, ...)`

Evaluation rules for call expressions

1. Evaluate the operator
2. Evaluate the operands from left to right
    - If an operand is a nested call expression, then these steps are applied to that inner operand first in order to evaluate the outer operand.
3. Apply the operator to operands

```
1  def f(x, y):
2      return x * 2 + y * 3
3
4  g = f
5  a, b = 4, 5
6  g(a, b)
7  # operator: g, which evaluates to the function defined above
8  #operands: a and b, which evalute to 4 and 5, respectively
```

`return` AND `print`

- `return`
  - Must be in function body
  - When Python executes a `return` statement, the function terminates immediately.
  - If Python reaches the end of the function body without executing a `return` statement, it will automatically return `None`
  - If a string is returned, the quote is preserved
- `print`
  - display values in the Terminal.
  - does not interfere with the execution flow of the function
  - If a string gets printed, `print` will display the string without quotes

return AND print - CONT.

```
>>> 6
6
>>> print(6)
6
```

```
def what_prints():
    print('Hello World!')
    return 'Exiting this function.'
    print('61A is awesome!')

>>> what_prints()
Hello World!
'Exiting this function.'
```

```
def what_prints():
    'Hello World!'
    return 'Exiting this function.'
    print('61A is awesome!')

>>> what_prints()
'Exiting this function.'
```

CONTROL 

BOOLEAN OPERATORS

- Some arithmetic expressions evaluate to boolean values

```
>>> 6 + 1 == 7
True
>>> 8 > 9
False
```

- Python boolean operators: `not`, `and`, and `or`
  - Priority: `not` > `and` > `or`
  - Use parenthesis to make your code more readable!

TRUTHINESS AND FALSINESS

Truthy Values

- Treated as practically "true" in boolean contexts (if/while conditions, and/or/not expressions)
- Everything that's not falsy is truthy

Falsy Values

- Treated as practically "false" in boolean contexts (if/while conditions, and/or/not expressions)
- Including: `0`, `None`, `""` (empty string), `[]` (empty list), etc.
- See here for a comprehensive list

SHORT CIRCUITING

- Short circuit - not every operand gets evaluated
- `and`
  - evalutes from left to right until the first FALSY value or the last value
  - return the last thing that's evaluated
- `or`
  - evalutes from left to right until the first TRUTHY value or the last value
  - return the last thing that's evaluated

```
>>> True and 1 / 0
ZeroDivisionError
>>> True or 1 / 0
True
```

```
>>> 1 and 2 and 3
3
>>> 1 or 2 or 3
1
```

CONDITIONAL STRUCTURES 

if STATEMENTS

```
if <cond_1>:
    <case_1>
elif <cond_2>:  # elif is optional
    <case_2>
else:  # else is also optional
    <case_3>
```

- Only one of the three cases will be executed.

```
if <cond_1>:
    <case_1>
if <cond_2>:
    <case_2>
if <cond_3>:
    <case_3>
```

- Here, more than one case could be executed.

if STATEMENTS - CONT.

```
if 3 > 2:
    print("hello")
elif "huh":  # elif is optional
    print("hi")
else:  # else is also optional
    print("QAQ")
# output
hi
```

```
if 3 > 2:
    print("hello")
if "huh":  # elif is optional
    print("hi")
if 0:  # else is also optional
    print("QAQ")
# output
hello
hi
```

while LOOPS

```
while <condition>:
    <do something>
```

- While `<condition>` is truthy, execute the body of the loop, and evaluate the condition again.

while LOOPS THAT EXECUTE k TIMES

```
count = 1
while count <= k: # pay attention to the equal sign here
    <do something>
    count += 1  # equivalent to count = count + 1
```

```
while k: # when k reaches 0, the condition becomes falsy
    <do something>
    k -= 1  # equivalent to k = k - 1
```

Both are useful templates to remember!

DIGIT MANIPULATION 

DIGIT MANIPULATION

- `num % 10` retrives the last digit of `num`
- `num // 10` removes the last digit of `num`
- Example: prints the digits of a number `n` in reverse order

```
while n:
    # isolate the last digit from n and print it
    print (n % 10)
    # removes the last digit from n
    n //= 10  # equivalent to n = n // 10
```

- Get started on the lab and raise your hand whenever you need help!
- Attendance: go.cs61a.org/mingxiao-att
  - By default, password will be released at the end of lab.
  - If you finish early, let me know and I'll check you off.
- Slides: go.cs61a.org/mingxiao-index