LABORATORY 2 : Simple Python Programming

OBJECTIVES

- to be familiar with Python IDE
- to know more about software testing
- to be able to write and test simple Python programs
- to write and understand a Python program that conforms to PEP-8

BACKGROUND

1. IDE

(http://en.wikipedia.org/wiki/Integrated_development_environment)

> An **integrated development environment** (**IDE**) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools and a debugger. Some IDEs, such as NetBeans and Eclipse, contain the necessary compiler, interpreter, or both; others, such as SharpDevelop and Lazarus, do not.

Python IDEs

There are many IDEs in the market that support Python. Some are free of charge, some are not. Some are open-source. IDEs usually are available in many versions for running on different platforms. One popular and stable Python IDE is PyCharm (http://www.jetbrains.com/pycharm/). However, many programmers prefer using Python console—IDLE or Notepad++.

2. PEP-8

PEP-8 is a style guide for Python code to improve the readability of Python code. It was created in 2001.

PEP-8 includes guideline for the following and more.

- naming conventions
- code layout
- indentation
- comments
- whitespace

Please refer to https://www.python.org/dev/peps/pep-0008/ or https://pep8.org/ for details.

3. Radix

(https://en.wikipedia.org/wiki/Radix)

> In a positional numeral system, the **radix** or **base** is the number of unique digits, including the digit zero, used to represent numbers. For example, for the decimal/denary system (the most common system in use today) the radix (base number) is ten, because it uses the ten digits from 0 through 9.

Examples of numeral systems:

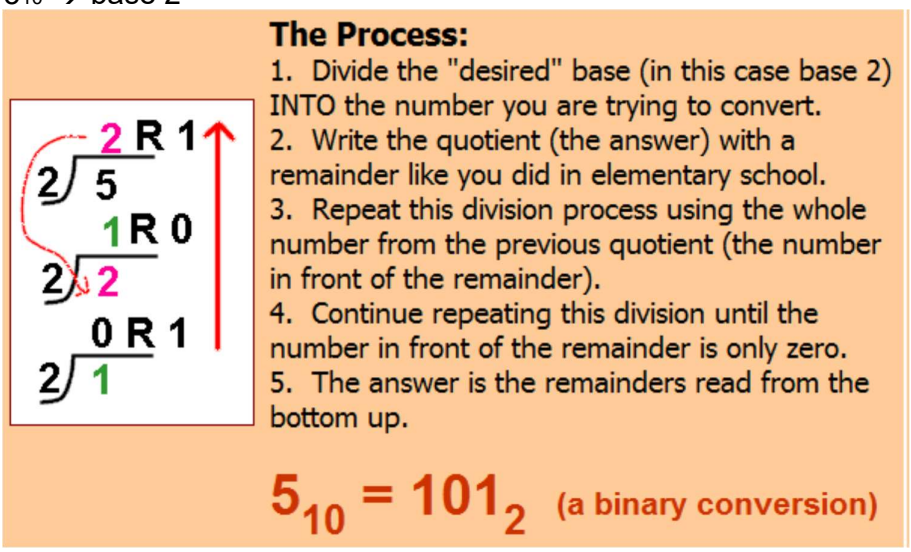
- The **decimal** system, the most used system of numbers in the world, is used in arithmetic. Its ten digits are "0–9".
- The **binary** numeral system, widely used in computing, is base two. The two digits are "0" and "1", representing different electric charges.
- The **octal** system, which is base 8, is also often used in computing. The eight digits are "0–7".
- Also in widespread use in computing is the **hexadecimal** system. It is base 16, and the 16 digits are "0–9" followed by "A–F".

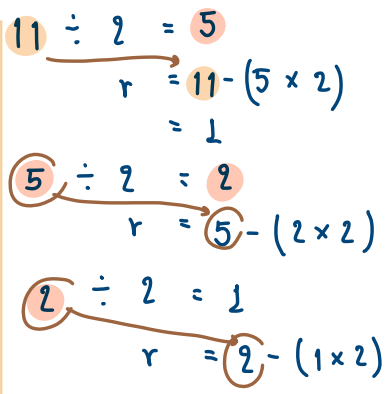
Since each position can hold only one character, alphabets (ABC…Z) are used to represent numbers that are greater than 9, where A represents 10, B represents 11, C represents 12, … , and Z represents 35.
Decimal number 10 is written in various bases as 1010_2, 22_4, 20_5 ,10_{10} and A_{11}

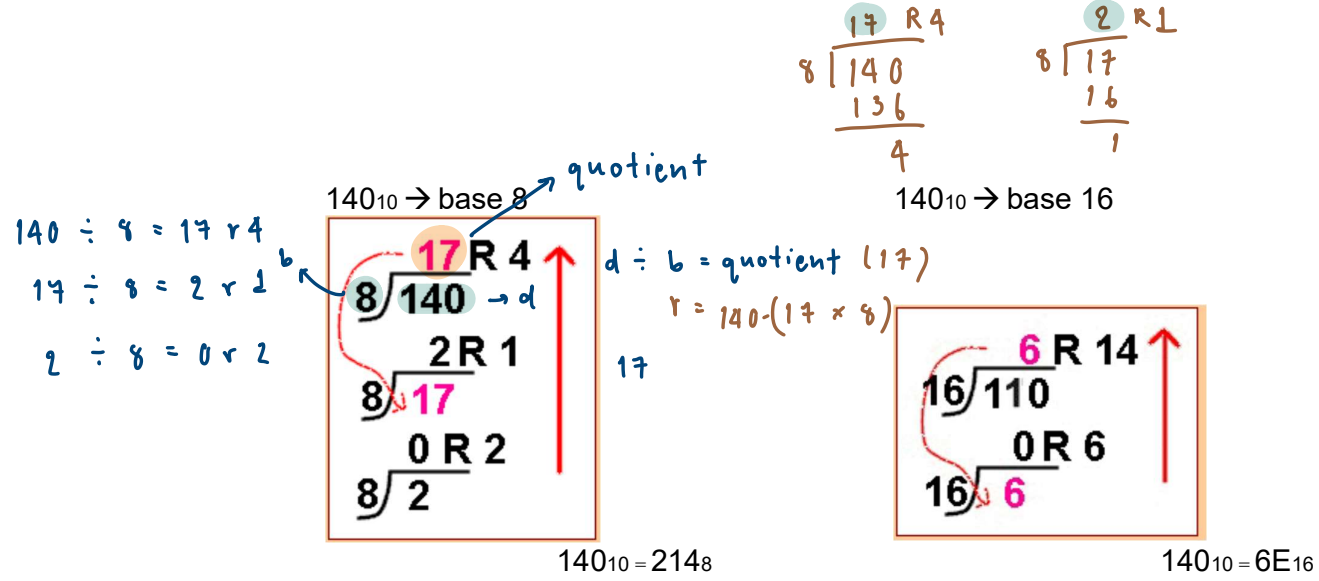
4. Number base conversion

- From decimal (decimal → other base)

5_{10} → base 2







140_{10} → base 8

$140_{10} = 214_8$

140_{10} → base 16

$140_{10} = 6E_{16}$

- From other base to decimal (other base → decimal)

235_8 → decimal

$8^2\ 8^1\ 8^0$
2 3 5

The Process:
Above each of the digits in your number, list the power of the base that the digit represents. See the example on the left. It is now a simple process of multiplication and addition to determine your base 10 number. In this example you have

$5 \times 8^0 = 5$
$3 \times 8^1 = 24$
$2 \times 8^2 = 128$

Now simply add these values together.
$5 + 24 + 128 = 157$
Answer: $235_8 = 157_{10}$

$1C4_{16}$ → decimal

$4 \times 16^0 = 4$
$C \times 16^1 = 12 \times 16^1 = 192$
$1 \times 16^2 = 256$

$4 + 192 + 256 = 452$
Answer: $1C4_{16} = 452_{10}$

5. Number base conversion functions

`from_decimal(d, b)`
converts a decimal number into a base b string number
Precondition : decimal number is any positive integer
base (b) is an integer greater than 1 and less than 37,
Postcondition : the given decimal number (d) is converted in to base b

`to_decimal(s, b)`
converts a string representing a base b integer into a decimal number
Precondition : each character in the string (s) must be between 0 and (when b is less than or equal to 9) or the alphabet represented by b (when b is greater than 9), base (b) is an integer greater than 1 and less than 37,
Postcondition : the given string (s) is converted in to a decimal number

LABORATORY 2: Pre-lab

1. Read “Python Primer” (Chapter 1, GTG’s) thoroughly. In case you don’t have that book, read your favorite Python book.
2. Download and install your IDE of choice.
   Make sure that you have everything required by the IDE before you begin installation. Pay attention to version numbers also.
3. Download and install Python API documentation on your machine. It’ll be helpful. (https://docs.python.org/3.8/index.html, Python 3.8)
4. Follow tutorial that comes with the IDE. Make sure that you know how to work around the IDE. For example, what all the panels are for and what information they contain, which button (or menu) to click if you want to debug a program, etc.

   For PyCharm, go to https://www.jetbrains.com/help/pycharm/quick-start-guide.html.
5. Find online tutorial on using debugger in your IDE. Follow the tutorial making yourself familiar with the debugger.

   (Debugging with PyCharm video can be found at https://www.youtube.com/watch?v=QJtWxm12Eo0)
6. Read about PEP-8 style guideline from https://www.python.org/dev/peps/pep-0008/. Make sure that you understand how to write a Python program that conforms to PEP-8.
7. Practice number base conversion on paper.

LABORATORY 2: In-lab, Post-lab

1. Follow a DataCamp’s PEP-8 tutorial on https://www.datacamp.com/community/tutorials/pep8-tutorial-python-code
2. Think and sketch up an algorithm to convert decimal number to any given base and vice versa. Write down your algorithms in a form of pseudocode or flowchart.
3. Design a test plan to test your number conversion program. You may start from the table below and write down more test cases on your own. Indicate type(s) of coverage given by your test cases.
4. Write number conversion functions according to the algorithm you wrote for 1. Document your code properly. Make sure that your code conforms to PEP-8. Note that you are not allowed to use Python’s int function.
5. Test your convertor following the test plan you wrote for 2.

Note: you may try using debugger. Set breakpoints and start the debugger from the Debug menu.

Submission:

via Canvas
submission details are posted in Canvas.
You are to review your work with the TAs during lab session.

Test plan for number base conversion

	Test case : Command and Input	Expected result	Path# or Line# that the test case covers

normal “from decimal” conversion	`from_decimal(250, 2)`
	`from_decimal(0, 10)`
	`from_decimal(12, 16)`
	`from_decimal(2553, 26)`
		`11111010`
		`0`
		`C`
		`3K5`
			depends on your algorithms → ←

incorrect number/base “from decimal” conversion	`from_decimal(-27, 2)`
	`from_decimal(15, -10)`
	`from_decimal(12, 27)`
		`Error :negative number`
		`Error :negative base`
		`Error :base > 36`	

normal “to decimal” conversion	`to_decimal(“100”, 2)`
	`to_decimal(“11”, 5)`
	`to_decimal(“A”, 16)`
	`to_decimal(“A”, 12)`
	`to_decimal(“989”, 10)`
	`to_decimal(“1BJ”, 26)`
		`4`
		`6`
		`10`
		`10`
		`989`
		`981`	

incorrect number/base “to decimal” conversion	`to_decimal(“56”, 3)`
	`to_decimal(“191”, 3)`
	`to_decimal(“-77”, 6)`
	`to_decimal(“22”, -9)`
	`to_decimal(“LOVE”, 37)`
		`Error: incorrect number/base`
		`Error: incorrect number/base`
		`Error: negative number`
		`Error: negative base`
		`Error: base > 36`	

...	...	...	...