I. INTERACTIVE MODE VERSUS SCRIPT MODE

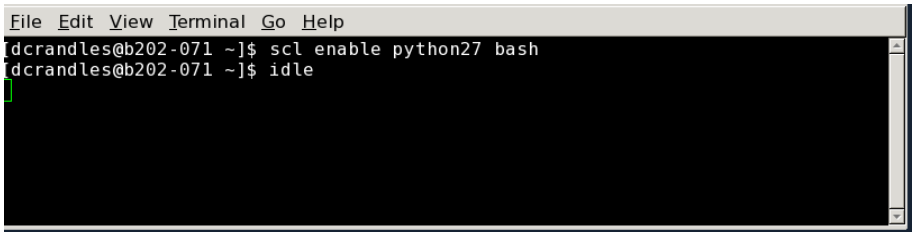
There are two ways to use the python interpreter: *interactive* mode and *script* mode.

1. **Interactive Mode**

   (a) open a terminal shell (terminal emulator in Applications Menu)

   (b) You may or may not have to type the following command: **scl enable python27 bash** $< CR >$ **(carriage return)** as shown in the figure below.

   (c) idle $< CR >$



   (d) This will open a new IDLE window (Interactive Development and Learning Environment). The chevron >>> is the *prompt* telling you that IDLE is ready to execute an instruction.

```
Python 2.7.13 (default, Feb  8 2017, 06:30:30)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-16)] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> 2+5
7
>>> 2-5
-3
>>> 2*5
10
>>> 2**5
32
>>> 2/5
0
>>> 2.0/5.0
0.4
>>> (2+5)**(5-2)
343
>>>
```

In the "IDLE conversation" shown above, you can see how the interpreter is

being used in interactive mode as a simple calculator: You type an instruction to the interpreter and after pressing $< CR >$, the instruction is carried out.

(e) The usual order of operations (PEDMAS) is followed by the interpreter. (Parenthesis-Exponentiation-Division-Multiplication-Addition-Subraction)

i. expressions in parenthesis are evaluated first, followed by the exponentiation operation.

ii. division and multiplication are evaluated next and have the same order.

iii. Operations of the same order are evaluated left to right.

iv. addition and subtraction are evaluated last and have the same order.

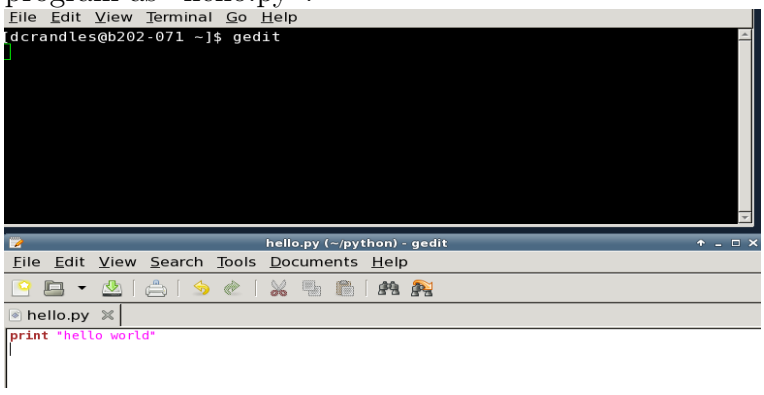
v. Note the difference in output between an instruction to divide two "integers" (2/5) and two "floats" (2.0/5.0). This is because python 2 performs *floor division* for integers and gets rid of digits after the decimal place. *Integers* and *floats* are two different *data types* which is a concept that will be discussed below. Python 3 outputs 2/5=0.4 and floor division is performed by a special operator.

vi. $< CTRL > D$ exits IDLE

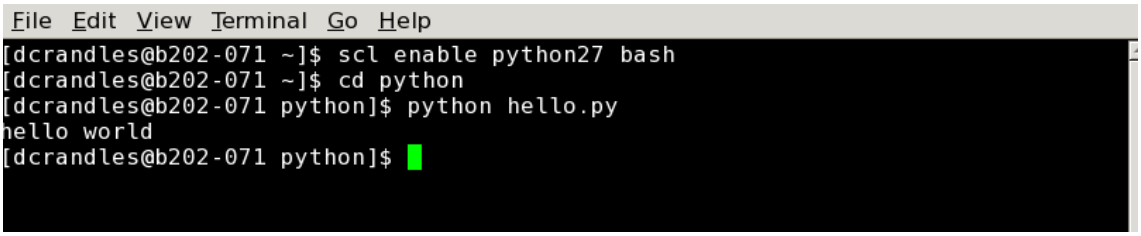
2. **Script Mode**

(a) You can also save your commands in a text file, which by convention have the suffix ".py", for example, program.py.

(b) Create your list of commands inside any text editor, for example gedit. Save this program as "hello.py".

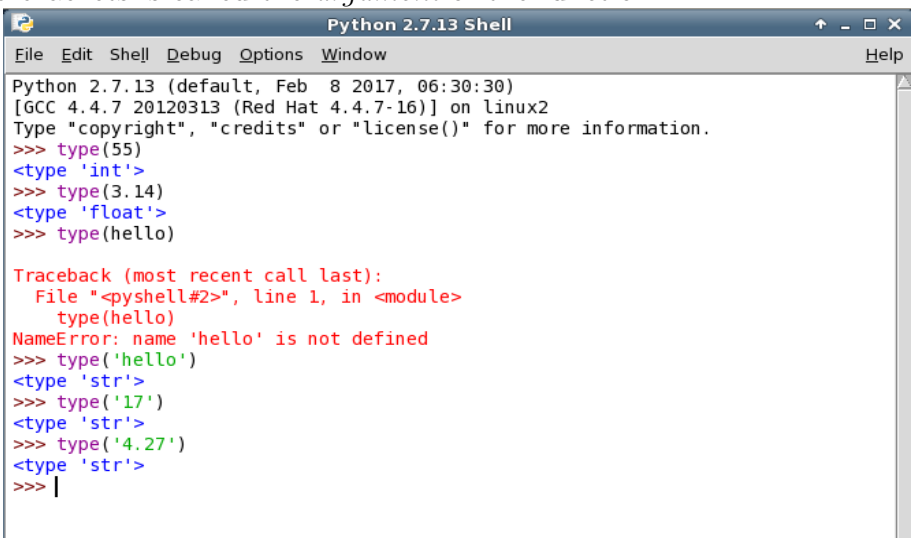


(c) run the program as follows (you have to be sure that you are in the directory where the script file "hello.py" is located)



II. VALUES AND TYPES

A *value*, such as 1,2, 3.14 or 'Hello World' is stored in a location in computer memory. The values belong to different *types*: (i) integer (e.g. 1,2,3), (ii) float (3.14) or (iii) string - so called because it contains a "string" of characters (e.g. 'Hello, World!'). Different types are stored in different ways in computer memory. The interpreter command **type()** can be used to determine type. **type()** is an example of a *function*. Whatever you type between the brackets is called the *argument* of the function.



Open IDLE and enter the commands shown above. Can you explain each python response? Can you explain the syntax error?

III. VARIABLES

A variable is a name given to a piece of computer memory. One reads the *assignment statement* **x=5.0** as meaning "place the float value 5.0 in memory location called x". One might also say "the state of the memory location that we are calling "x" is the float value

"5.0" . Recreate the following IDLE conversation where values are assigned to the variables n, e and $message$:

```
>>>
>>> message='you are an allstar'
>>> n=15
>>> e=2.718281828459045235360
>>> type(message)
<type 'str'>
>>> type(n)
<type 'int'>
>>> type(e)
<type 'float'>
>>>
```

One can see that the variable type is determined by the assignment statement. If you recreate IDLE conversation below

```
>>>
>>> e=2.718281828459045235360
>>> type(e)
<type 'float'>
>>> e=17
>>> type(e)
<type 'int'>
>>> e=3.141593
>>> type(e)
<type 'float'>
>>> e='you are beautiful'
>>> type(e)
<type 'str'>
>>>
```

one will see that the type of a particular variable is not fixed, but can be changed by an assignment statement.

IV. VARIABLE NAMES AND KEYWORDS

A programmer should strive to use meaningful names that enhance the readability of a program. Variable Names can be arbitrarily long. They can contain both letters and numbers and are case-sensitive, but they must begin with a letter. That is $ArKiv$ and $arkiv$ are different variable names. To make the names meaningful, consider using underscores for very long names such as *speed_of_a_bullet* to improve readability of program. If underscores are not used, you will get a syntax error . For example, *fun fact = 100.0* is illegal but *fun_fact=100.0* is legal.

Consider the following IDLE conversation containing three illegal names illustrating that variable names must start with a letter, cannot contain illegal symbols (@,$ etc.) and cannot be one of Python 2's 31 keywords. The interpreter uses the keywords to recognize the structure of the program and hence they cannot be used as variable names. in Python 3, "exec" is no longer a keyword but "nonlocal" is.

```
>>>
>>> 99red='balloons'
SyntaxError: invalid syntax
>>> more$=4000
SyntaxError: invalid syntax
>>> lambda='point'
SyntaxError: invalid syntax
>>>
```

The list of python 2 keywords is as follows:

and	**del**	**from**	**not**	**while**
as	**elif**	**global**	**or**	**with**
assert	**else**	**if**	**pass**	**yield**
break	**else**	**if**	**pass**	
class	**exec**	**in**	**raise**	
continue	**finally**	**is**	**return**	
def	**for**	**lambda**	**try**	

V. COMMENTS

Obviously, complicated programs are run in script mode. As programs get larger and more complicated, they get more difficult to read. It is a good idea to add notes to explain what the program is doing, and non-obvious features of the code. This is true if more than one programmer is working on a project, or if a single programmer is writing a long program. It is very often difficult to recall exactly what your program is doing. Called *comments*, these notes start with the '#' symbol. Anything after the comment symbol in a line is ignored by the interpreter.

```
# the following is a useless comment because it is obvious

area= 5 # assign 5 to the variable area

# the following is a good comment because it provides useful information not in the code.

area = 5 # area in square meters
```

VI. EXERCISES I

These exercises are copied from "Think Python", by Allen Downey.

1. Start IDLE and type help() to start the online help utility to get information about the *print* statement.

2. Start IDLE and use it as a calculator:

   (a) to determine the average speed (in km/h) of Sir Roger Bannister who, in 1954, was the first person to run a mile in under four minutes (3 minutes, 58 seconds).

   (b) to determine the number of seconds in a year.

   (c) Determine the volume of a sphere $\frac{4}{3}\pi r^3$ of radius 7 cm. Hint: Using $\pi = 3.1416$, the value 1077.5688 is wrong.

3. Write a program using *gedit* or another text editor that will print a well-structured English sentence with invalid tokens in it and run it in script mode. (See the introduction for definition of token)

4. Write a program using *gedit* or another text editor that will print an English sentence with valid tokens but with an invalid structure and and run it in script mode. (See the introduction for definitions of tokens, structure and syntax)

5. What is the problem in the following IDLE conversation? Is this a semantic error or a syntax error?

```
>>>
>>> 5
5
>>> 1,000,000
(1, 0, 0)
>>>
```

6. If you begin an assignment statement with a leading zero, you might get confusing errors.

```
>>>
>>> zipcode=02492
SyntaxError: invalid token
>>> zipcode=02132
>>> print zipcode
1114
```

Can you figure out what is going on? Hint: display the values of 01, 010,0100,01000.