Lecture 1: Intro. to Programming with Python*

*Partially based on Chapter1 and 2 of Think Python: How to Think Like a Computer Scientist (see Reading List)

- Problem Solving & Programs
- Values, Types and Variables
- Assignment Statements
- Arithmetic Operators, Order of Operations
- Program Comments
- Introduction to Strings
- print(), str()

Problem Solving & Programs

Problem solving means the ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately.

Problem solving is an important skill for a computer scientist and learning to program will be an opportunity to practice problem-solving skills.

Problem Solving & Programs (Cont’d)

- **A Program -** sequence of instructions that specifies how to perform a computation. Details look different in different languages, but basic instructions appear in most languages.
  - *input:* Get data from the keyboard/file/database etc.
  - *output:* Display on the screen, save to file/database etc.
  - *math:* Perform basic mathematical operations.
  - *conditional execution:* Check for condition and run appropriate code.
  - *repetition:* Perform some action repeatedly
- Programming - process of breaking a complex task into smaller & smaller subtasks until simple enough to be performed with one of the above.

Values and Types

- Programs work with **values.** These values belong to different **types**:
  - 2 is an **integer** - whole numbers (4, 99, 0, -99)
  - 42.0 is a **floating-point number** with decimal points (3.5, 42.1)
  - 'Hello World!' is a **string** (allows single/double quotes “Hello”)
  - True or False is a **boolean** variable
- Python supports *integer*s and *floating-point* numbers. There is no type declaration to distinguish them; Python tells them apart by the presence or absence of a *decimal* point.
- Question - How does Python tell the difference between a string and boolean variable if the type is not declared?

Variables (Cont'd)

- Different languages have different naming conventions for variables (e.g., **camelCase)**. For Python it is recommended to use lowercase with multiple words separated with underscores:

```
your_name
airspeed_of_unladen_swallow
```

- Cannot be a Python 3 reserved keyword:

```
'False', 'None', 'True', 'and', 'as', 'assert', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in',
'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
'return', 'try', 'while', 'with', 'yield'
```

Variables

- A name that refers to a value.
- The value stored in a variable can be accessed or updated later.
- Programmers should choose descriptive variable names.
- Should follow the following **variable names rules**
  - Must begin with a letter (a - z, A - Z) or underscore (_).
    - Other characters can be letters, numbers or _
  - Variable names are case sensitive. These are different variables:
    case_sensitive, CASE_SENSITIVE, Case_Sensitive
- Must not have spaces inside them (e.g., 'running total' not allowed).

Self-Check 1

Which of these are illegal variable names?

a. the cost
b. 2_much
c. much2
d. *star
e. more@
f. class
g. the_cost

Assignment Statements

- An **assignment statement** creates a variable and gives it a value:

```
message = 'Something completely different'
n = 17
pi = 3.141592653589793
```

- The first assigns a string to a variable named message;
- The second assigns the integer 17 to variable n;
- The third assigns the (approximate) value of *π* to variable pi.

A statement

A **statement** is a unit of code that has an effect:

```
n = 17
print(n)
```

- The first line is an assignment statement that gives a value to n.
- The second line is a print statement that displays the value of n.
- You can change the value stored in a variable by entering another assignment statement.

```
n = 6
```

- The previous value 17 is replaced, or overwritten with the value 6.

Self-Check 2

Set variables a = 1 and b = 2.

Then write the instructions to swap them so the value in a ends up in b and the value in b ends up in a.

Arithmetic Operators

Operator	Operation	Examples
+	Addition	b = a + a
-	Subtraction	newTotal = price – discount
*	Multiplication	total = cost * vat
/	Division Python 3 - integer / integer -> float	8 / 3 # 2.6666666666666665
%	Modulus - Returns remainder	result = 16 % 5
**	Exponent	answer = 4**2 (4 to power of 2)
//	Floor division (only integer part) Python 3 - integer // integer -> integer	8 // 3 # 2

Note: If a program tries to divide by zero, the program is terminated/produces error

Program Comments

- As programs get more complicated add comments to your programs to explain what the program is doing. Python uses # symbol.
- Everything from the # to the end of the line is ignored.

```
percentage = (minute * 100) / 60       # hour elapsed
```

- Comments can appear on a line by itself.

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

- Comments most useful when they document non-obvious code features.
- Redundant comment: `v = 5    # assign 5 to v`

Self-Check Exercise 3

3. What will be the values in the variables at the end of each program sequences? On paper draw a box for each variable, and show the changing values (cross out old values & write new values).

```
    x = 3
    y = 5
    z = 6
    x = y + z + x                    # value of x?          _________
----------------------------------------------------------------------
    a = 2
    b = 3
    a = b + a
    b = a + a                        # value of b?          __________
```

Self-Check Exercise 4

```
cost = 5
factor = 7
factor = factor * 3
factor = factor + cost            # value of factor?  __________
```

```
cost = 10
vat = 17.5
total = cost * vat                # value of total?  __________
```

Self-Check Exercise 5

```
a = 2
b = 3
c = a**b                                # value of c?          __________
--------------------------------------------------------------------------
num_1 = 21
num_2 = 10
num_3 = num_1/num_2                     # value of num_3?      __________
--------------------------------------------------------------------------
num_4 = num_1//num_2                    # value of num_4?      __________
```

Self-Check Exercises 6 & 7

6. Write the code to put 4 into a variable called 'item1'. Then put 6 into a variable called 'item2'. Then write an instruction to add the two variable values together and put the answer into a variable ‘item3‘.

7. A meal costs £56. Write the code to set 56 into a variable. Then multiply whatever is in the variable by 1/10 to work out the 10% tip (store the answer in a variable).

Order of Operations

When an expression contains more than one operator, the order of evaluation depends on the **order of operations**.

Acronym **PEMDAS -** useful to remember rules:

- **P**arentheses have the highest precedence & can force evaluation in the order required:

```
2 * (3-1) is 4
```

- You can use parentheses to make an expression easier to read (doesn’t change result):

```
(minute * 100) / 60
```

Order of Operations (Cont'd)

- **E**xponentiation has the next highest precedence:

```
1 + 2**3                # is 9, not 27
2 * 3**2                # is 18, not 36.
```

- **M**ultiplication and **D**ivision have higher precedence than **A**ddition and **S**ubtraction:

```
2 * 3 - 1               # is 5, not 4
6 + 4 / 2               #  is 8.0, not 5.0.
```

- Operators with the same precedence are evaluated left to right.

```
degrees / 2 * pi
# division first and then multiply by pi
```

Self-Check Exercise 8

- Using the acronym PEMDAS, what is the value of the following expressions?

```
2**1+1                # result is 3 or 4?

3*1**3                # result is 3 or 27?

16 - 2 * 5 // 3 + 1   # result is 14, 24, 3 or 13.667?
```

Introduction to Strings

- A string is a sequence of characters.
- Python allows single ('...') or double quotes ("...") to surround strings.

String Operations

- In general, you can’t perform mathematical operations on strings:

```
'2'-'1'                   # illegal
```

- Two exceptions follow.
  - The + operator performs **string concatenation.** E.g.,

```
first = 'throat'
second = 'warbler'
third = first + second    # throatwarbler
```

  - The * operator performs repetition on strings.

```
'Spam' * 3:               # SpamSpamSpam
```

Strings – Introduction

.

If a single quote is a part of the string place string in double quotes.

- Double quoted strings can contain single quotes inside them:

```
"Bruce's beard"    # not 'Bruce's beard'
```

- Single quoted strings can have double quotes inside them:

```
'She said "Hi!"'   # not "She said "Hi!" "
```

- Using escape sequence (\") or (\'):

```
'Bruce\'s beard'
```

Strings – Introduction

Printing strings over multiple lines using triple-quotes:

```
hello = '''This is one line.
Another line.'''
print(hello)
```

Using escape sequence (\n) - printing strings over multiple lines:

```
print('This is one line.\n Another line.')
```

Common escape sequences:

Sequence	Meaning
\\	literal backslash
\'	single quote
\"	double quote
\n	Newline
\t	Tab

Lecture Self-Check Question

Assign a variable **question** with the value:

Where’s the lecture room?

Built-in Functions

- A function is a piece of code written to carry out a specified task.
- To use an existing built-in function, you need to know its name, inputs and outputs.
- **print() function -** sends content to the screen
- Python is case sensitive. Use print(), rather than Print() or PRINT().

```
print()                       # empty line
print('Hello')                # Hello
print(42)                     # 42
greeting = 'Hello'            # Assign a string to variable
print(greeting)               # print variable value
```

print() function

- Multiple objects separated by commas print separated by a space:

```
print('dog', 'cat', 42)
# dog cat 42
```

- To suppress printing of a new line, use end='' :

```
print('Dog', end='')
print('Cat')
# DogCat
```

print() with string concatenation

```
product = 'mask'
price = 99
print("The product: " + product + "costs" + str(price) + "pence")
```

- String concatenation requires strings! Convert price with str().

```
The product: maskcosts99pence
```

- The + operation on strings adds no extra space between strings.

```
print("The product: " + product + " costs " + str(price) + " pence")
```

print() with commas (reminder)

```
product = 'mask'
price = 99
print("The product:", product, "costs", price, "pence")
```

- The 5 arguments passed to print will be **converted to strings** and **with a space between**:

```
The product: mask costs 99 pence
```

Lecture 1

- Problem Solving & Programs
- Values, Types and Variables
- Assignment Statements
- Arithmetic Operators, Order of Operations (**PEMDAS**)
- Program Comments #
- Introduction to Strings
- print(), str()