Introduction to:
Computers & Programming:
Print Statements, Data Types, Functions, Variables

Adam Meyers
New York University
Outline

• Print Statements in Python
  – The 'Hello World' Program
  – Print is a function that takes a string as an argument
• Data types: strings, integers, floats, etc.
• Introducing Functions, Operators and Variables
Hello World

• Custom when learning programming language
• Write a simple program that causes the string 'Hello World' to be printed to the screen.
• In Python 3
  – print('Hello World ')
• Syntax of print
  – All-lowercase “print” followed by
  – Parentheses containing what is to be printed
  – Strings to be printed are surrounded by quote marks
  – Multiple items to print can be separated by commas
Now let's do it wrong

• Suppose we don't use quotes
  – Invalid syntax
  – One can only print legitimate Python objects
    • Such as strings (others will be discussed)

• Suppose we use double quotes
  – In Python, quotes, double quotes or even triple quotes are allowed
  – But they have to match

• Suppose we capitalize the first letter in “Print”
  – Lowercase “print” is the name of a function
  – Capitalized “Print” is not the name of a function
What is a variable?

- A named object that has a value
- Typically it can “vary”
- A program for computing shoe size
  - May use a variable that equals the size of a foot
  - It will calculate a different shoe size depending on this foot size
- `=` assigns values to variables in Python
  - Example: `footsize = 12 (inches)`
- Variable names clarify what a program is doing
  - `footsize` is a good variable name
  - `x` is a mediocre variable name
- Using “magic numbers” or other hard-coded information
  - makes programs difficult to understand or modify
Examples

• def clear_farenheit_to_centigrade(farenheit):
  multiple = 5/9
  difference = 32
  centigrade = (farenheit-difference)*multiple
  return(centigrade)

• def confusing_f_to_c(f):
  return((f-32)*(5/9))
What is a function in math?

- A mapping from each element in one set (the domain) to exactly one element in another set (the range)

Examples

- The square of a number
  - Given any number (the domain), there is exactly one square in the range, a subset of the set of numbers

- The shoe size of a human being
  - For each member of the human race, there is exactly one shoe size (ignoring differences between brands, asymmetric left/right feet, etc.)

- In Python (and other languages), a function is not exactly the same thing, but sort of
Functions, Procedures, Subroutines, Methods, etc.

- In Programming, all these terms are used to describe essentially a command, defined in terms of a set of statements of that language.

- A Function can provide a mapping from input (domain) to output (range)
  - Like the mathematical definition
  - Input can be a set of zero or more items
    - These are called input parameters
  - Output can be a set of zero or more items
    - These are preceded by the key word `return`

- In programming, functions can have side effects
  - Stuff that happens before output is returned, e.g.,
    - Communication with humans, e.g., helpful print-outs, sounds, images
    - Bookkeeping processes that effect other parts of complex program
In Python 3, “print” is a function

- **Input**: zero or more objects as input
  - Executing with no input causes a newline to be printed
- **Output**: None (a special object)
  - Means something like 'no value'
  - Programmers don't normally use the output of 'print'
  - Sometimes, such functions are called *void functions*
- **Side effect**
  - It prints out (string versions of) the input
    - In IDLE, it prints them on a line
    - The side effect is the important feature of this function
- **Main purpose**: print stuff to the screen for human user to read
“input” is also a Python 3 function

- **Example:**
  - `input('Give me a Number')`
  - If the user enters 5, it returns the string '5'

- **Input:** 1 argument, a prompt string
- **What it Does:** pauses until user types something and hits return
- **Output:** string consisting of what the user types
- **Purpose:**
  - To solicit user input, when program interacts with user
  - Most functions do NOT contain input statements
  - Use only for interactive portion of programs
Writing a Function in Python

• `def function_name (arg1, arg2, ...):`  # name & parameters
  stuff  ## statements that are executed
  return(output)  ## returning a value

• `def add_two_numbers(number1, number2):`
  print('Adding',number1,'and',number2)
  sum = number1 + number2
  print('The result is', sum)
  return(sum)

• *number1* and *number2* are input parameters
• The 2 print statements produce side effects
• *return* returns output (the sum of number1 and number2)
Using input statements with add_two_numbers

- def interactive_add_2_numbers():
  
  print('Please supply 2 numbers.')
  first_num = input('Give me a number. ')
  first_num = float(first_num)
  ## float converts string to floating point number
  second_num = input('Give me another number. ')
  second_num = float(second_num)
  answer = add_two_numbers(first_num, second_num)
  return(answer)
Using values returned by functions

• return gives “value” to function calls

• `add_two_numbers(5,5) + add_two_numbers(5,5)`
  – returns 20
  – assume statement “`add_two_numbers(5,5)`” means 10

• `output = add_two_numbers(5,5)`
  – assigns the variable `output`, the value 10

• In contrast, `print` prints something to the screen
  – can't use `print_two_numbers` like `add_two_numbers`
    • `def print_two_numbers(num1,num2):
      print(num1+num2)"
Data Types

• There is a function “type”
  – An object of any type is the input argument
  – The **data type** is the output

• Use this function in IDLE as follows:
  • `type('Hello')`
  • `type("Hello")`
  • `type(5)`
  • `type(5.4)`
  • `type(print('hello'))` ## Notice the side effect of this statement

• **Built in data types:**
  • Numeric types: integer, float, complex
  • Sequence types: string, list, tuples
  • Other: Boolean, Dictionary, Set, NoneType, ...
Strings in Python

• A string is a sequence of characters surrounded by
  • Single quotes 'string'
  • Double quotes “string”
  • Triple quotes """string"

• Embedding 1 kind of quote within another
  • 'the string “Python 3.1” labels this language'
  • “the string 'Python 3.1' labels this language”

• Newlines can be embedded in triple quotes
  • ""These two strings, 'Python 3.1' and
    “Python 3.1”, are on separate lines""
Numeric Data Types

- **Integer**
  - Whole numbers (positive and negative)
  - -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5

- **Float (decimal numbers)**
  - -2.305, -0.1334, 0.33333333, 14.789, 5.0

- **Many-digit floats are shortened (all programming languages do this since floats can be infinitely long).**
  - Long decimals are truncated, rounded or written in scientific notation
  - Try: .11111111111111111111119 and 1111111111111111111.9
  - Takes more bytes to represent longer numbers, but unnecessary for most purposes
  - Systems support high precision arithmetic for applications requiring higher precision (https://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic)

- **Complex: sum of floats and multiples of imaginary numbers**
  - \( j = \sqrt{-1} \)
  - 4.1+3j, 1+45j, …
### Operators are Functions with the Syntax of Arithmetic

<table>
<thead>
<tr>
<th>Operation</th>
<th>Standard Arithmetic</th>
<th>Python</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>$5 + 3$</td>
<td>5+3</td>
<td>8</td>
</tr>
<tr>
<td>Subtraction</td>
<td>$5 - 3$</td>
<td>5-3</td>
<td>2</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$5 \times 3$ or $5 \times 3$</td>
<td>5*3</td>
<td>15</td>
</tr>
<tr>
<td>Division</td>
<td>$5 \div 3$ or $5 / 3$</td>
<td>5/3</td>
<td>1.6666666</td>
</tr>
<tr>
<td>Exponents</td>
<td>$5^3$</td>
<td>5**3</td>
<td>225</td>
</tr>
<tr>
<td>Integer Division</td>
<td>$5 \div 3$ or $5 / 3$</td>
<td>5//3</td>
<td>1</td>
</tr>
<tr>
<td>Modulus</td>
<td>$5 \mod 3$</td>
<td>5%3</td>
<td>2</td>
</tr>
</tbody>
</table>
Operator Syntax vs. Function Syntax

• Each operator takes 2 arguments
  – Syntax of Function:   F(arg1, arg2)
  – Syntax of Operator:    arg1 Op arg2

• Python (and other languages) adopt this syntax because it is intuitive due to our education

• Other languages like LISP adopt Polish Notation
  – (+ 5 3), (-5 3), (* 5 3), (expt 5 3), (mod 5 3)
  – In such languages, there are no operators, only functions
Some Math from Grade School You May Have Forgotten About

• Grade School Division
  – $5 \div 3 \rightarrow 1$ remainder $2$

• Two Separate Operations in Python
  – Integer Division: $5\//3 \rightarrow 1$
  – Modulus: $5\%3 \rightarrow 2$

• Order of Operations: PEMDAS
  – Parentheses, Exponents, Multiplication/Division, Addition/Subtraction
  – Does $2+5*2$ equal $12$ or $14$?
Ambiguity without PEMDAS

\[
\begin{align*}
\text{Left:} & \quad 2 + 5 \times 2 \\
\text{Right:} & \quad 2 + 5 \times 2
\end{align*}
\]
Ambiguity gets worse as expressions get longer
Use Parentheses instead of PEMDAS

• Parentheses eliminate ambiguity from arithmetic
  – (2*5)+2 ==12
  – 2 * (5+2) == 14
  – (2**(5+2)) * 7 == 896
  – 2**((5+2) * 7) == 562949953421312

• The PEMDAS defaults also eliminate ambiguity
  – But for computers, not for most people

• All operators have this same ambiguity
  – Examples: %, and, or, in, =, ==, >=, >, <, <=, !=
The Modulus Operator

• Modulus gets the remainder from division
  – 100 % 12 == 4
  – 52 % 12 == 4
  – 28 % 12 == 4

• Example Application
  – Musical Instrument Digital Interface (MIDI)
    • MIDI notes begin with 0 (4 octaves below middle C)
    • MIDI notes end with 127 (G about 5 ½ octaves above middle C)
    • The statement 'MIDI-NUMBER % 12' identifies note classes
      • 0, 12, 24, 36, 48, 60, 72, 84, 96, 108, 120 are all C
      • 1, 13, 25, 37, 49, 61, 73, 85, 97, 109, 121 are all C#
      • 2, 14, 26, 38, 50, 62, 74, 86, 98, 110, 122 are all D
      • Etcetera ...
Operations on Strings (and 1 function)

- You can “add” (i.e., concatenate) strings
  - 'hello' + 'hello' == 'hellohello'
  - 'hello' + ' ' + 'hello' == 'hello hello'

- You can multiply (repeat) strings by an integer
  - 'hello' * 3 == 'hellohellohello'
  - 'hello ' * 3 + 'hello' == 'hello hello hello hello hello'
  - ('hello ' * 3) + 'hello' == 'hello hello hello hello hello'
  - What about 'hello ' * (3 + 'hello')?

- You cannot use / or ** with strings

- len('spam') – function indicating length of string
Conversion to Type String

• The `str` function converts objects to strings
  – `str(5)`
  – `str(5.1)`
  – `str(print('hi'))`
  – Any Python object can be converted to a string
    • This depends on the definition of that type of object

• The print function converts objects to strings before printing them
  – This also depends on the object's definition
Printing Non-strings

• For numbers, this is pretty obvious (try it)
  – print(5, 6, 7) ## printing integers
  – print(5.5, 100.1) ## printing floats

• For other objects, this is not so obvious
  – Print(1+ 5j) ## it adds ( ) around complex numbers
  – print(print(5)) ## prints the 5 as a side effect and then
    ## prints the output of print as 'None'

• Definitions of types of objects can include printing
  instructions, how they are represented as a string, and
  many other non-obvious properties
Comments

- A comment is a statement that is not evaluated
- In Python (and many languages)
  - Everything following “#” is interpreted as a comment
  - For example,
    
    ```python
    print('Hello World')  # This will print Hello World
    ```
  - Comments are used by programmers to help make their code readable (by themselves, by others, etc.)
  - Comments can occur after code or on lines by themselves
    
    ```python
    ## This is a comment – it will do nothing
    ```
- Multi-line comments take the form of strings, i.e.,
  - """This is a multi-line comment. This is the first line and this is the second"""
Multi-line Code

• The backslash at the end of a line indicates that the next line “counts” as being part of the current line, e.g.,

• def do_something():
      print('blah','blah','blah','blah','blah','blah','blah',\
            'blah','blah','blah')

• As if all the instances of 'blah' are on 1 line.
Conversion to Integers and floats

- The `int` function converts floats and some strings to integers
  - `int(5.1)` # OK
  - `int('5')` # OK
  - `int('five')` # will cause an error

- The `float` function converts integers and some strings to floats
  - `float(5)` # OK
  - `float('5.1')` # OK
  - `float('5')` # OK
  - `float('five')` # will cause error
Other Python Functions/Operators
Convert Integers to Floats

• “Normal” division
  – 4/2 == 2.0
  – 5/2 == 2.5
  – 1/3 == 0.3333333333333333

• But Not Integer Division
  – 4//2 == 2
  – 5//2 == 2

• Some of the functions in the math module
  – math.ceil, math.floor, math.trunc
Some Rules for Names and Variables

• Variables and Functions have Names
• Rules guiding Names
  – They can consist of letters, numbers and underscore __
  – They cannot begin with a number
  – They cannot be a special word reserved by Python:
    • and del from not while as elif global or with assert else if pass
      yield break except import print class exec in raise continue
      finally is return def for lambda try
• Be careful not to use the same name for different variables
• Scope
  – Parameters and variable defined inside a function, only work in that
    function – these are both instances of local variables
  – Variables defined outside functions are called global variables
  – Warning: Do not use the same name for a global variable and a
    local variable: we will discuss this more later in the term
Summary 1

• Functions map input to output and sometimes have side effects
  – This is similar to the mathematical concept of a function
  – Operators are like functions, but they have different syntax

• The Python 3 function 'print' displays strings as a side effect, the output of the function is unimportant

• Using functions and operators correctly requires using the proper syntax (spelling, punctuation, indentation, …)
Summary 2

• The print function prints the string version of an object as a side effect

• Types are categories of objects
  – An operator/function may handle different types differently
    • 'Hello ' + 'World' vs. 5 + 2
  – Some operators/functions are only defined for certain types
    • ** and / are defined for integers and floats, not strings

• Programming languages (and math) employ rules of precedence to make sure that statements are unambiguous
  – Such rules, e.g., PEMDAS, sometimes make it difficult for human beings.
  – However, parentheses can make programs readable for both humans and computers
HW 2

• Go to HW part of website:
  – http://cs.nyu.edu/courses/fall17/CSCI-UA.0002-004/#Homework%20Schedule
  – http://cs.nyu.edu/courses/fall17/CSCI-UA.0002-011/#Homework%20Schedule

• See all items due September 20
  – Module 3, Chapter 3, Quiz 3, Assignment 2