CISC101 Reminders & Notes

- Test 1 next week in your tutorial
  - Informal review today
- Are you not in the lab/tutorial section(s) you want?
  - Contact Irene LaFleche (`irene@cs.queensu.ca`)
- Assignment 1 due on Sunday by midnight
  - Updates to description
  - One change to requirements
  - Make sure your submission meets the requirements!
    - Read the description carefully

Slides courtesy of Dr. Alan McLeod

Today

- Introduction to …
  - Console I/O
  - Functions
  - Variable scope
  - Style
- We'll see more on some these topics later
  - Cover the basics for now

Slides courtesy of Dr. Alan McLeod

Console (or Screen) I/O

- Where "I/O" stands for "Input/Output"
- Output: use the `print(…)` BIF
  - Does not return anything
- Input: use the `input(…)` BIF
  - Returns a string
- Format: use the `format(…)` string function
  - Returns a string

Slides courtesy of Dr. Alan McLeod

The `print(…)` BIF

```
print(string1, string2, ...,
      sep=sepString, end=endString)
```

- Prints the string arguments given
- Separates the arguments with *`sepString`*
  - **`sep`** is a space unless otherwise specified
- Ends the printed sequence with *`endString`*
  - **`end`** is `\n` unless otherwise specified

Slides courtesy of Dr. Alan McLeod

The `input(…)` BIF

`input``(`*`promptString`*`)`

- Displays *`promptString`* on the screen
- Awaits keyboard input by the user
  - User presses Enter to end input
- Returns the entered data as a string
  - Must convert to another data type if necessary

Slides courtesy of Dr. Alan McLeod

The `format(…)` String Function

`aString``.`**`format`**`(`*`arg1, arg2, ...`*`)`

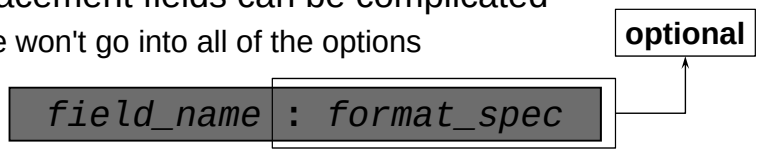
- Copies *`aString`* and inserts the given values
- *`aString`* uses *replacement fields* to represent the arguments
  - Each field is replaced by the specified argument
  - Fields also specify the format of the argument
  - Fields are always surrounded by `{}`

```
"What is {0} + {1}?".format(2, 5.0)
        → 'What is 2 + 5.0?'
```


Slides courtesy of Dr. Alan McLeod

Replacement Fields

- Replacement fields can be complicated
  - We won't go into all of the options



- *`field_name`*s specify arguments in two ways
  - Via indexes (starting from 0)
    - `"My name is {0}, {1} {0}".format("Bond", "James") → 'My name is Bond, James Bond'`
  - Via identifiers that you name and assign values to
    - `"{numer}/{denom}".format(numer=2, denom=4) → '2/4'`

Slides courtesy of Dr. Alan McLeod

Format Specifications

- Format specifications can be very complicated
  - We won't go into all of the options

`0 width .precision type`

- All of the above are optional
  - You can use one, some or all of them
- *`0`*: displays leading zeroes for numbers
- *`width`*: sets the minimum space occupied by the data

Slides courtesy of Dr. Alan McLeod

Format Specifications – Cont.

- *precision*: number of digits after decimal point
  - Specify *type* `f` for floating point
- *type*: dictates how the data is presented
  - `f` specifies a float
  - `b`, `o` and `x` convert arguments to binary, octal and hex
  - `s` for string is optional
- Some combinations cause errors
  - *e.g.*, you can't give a *precision* for "Hello"

Slides courtesy of Dr. Alan McLeod

What is a Function?

- A "group of statements" that accomplish a task
  - Perhaps composed of several smaller tasks …
- A function contains code that is *isolated*
  - Interacts with other code through a designed *interface*
- The interface consists of …
  - parameters for values that go into a function
  - return value(s) that comes out
- You can have as many parameters as you want
  - Including none
- You can return nothing or a single thing
  - Or more than one thing (as we'll see later)

Slides courtesy of Dr. Alan McLeod

Invoking Functions

- Name the function and then use round brackets
- Brackets contain zero or more *arguments*
  - Values for the function's parameters
    - "Parameter" and "argument" are often used interchangeably
- For example:
  - **`print()`**
    - Displays a linefeed on the console
  - **`print("Hello")`**
    - Displays the string Hello on the console
  - **`print("Hello", "Alan")`**
    - **D**isplays Hello and then Alan separated by a space

Slides courtesy of Dr. Alan McLeod

Invoking Functions - Cont.

- Arguments are separated by commas
- Arguments can be
  - Literal values
  - Variables
  - Expressions
- Variables and expressions are evaluated first
  - Determine the resulting value before invoking
  - Feed it into the function

Slides courtesy of Dr. Alan McLeod

Writing Functions

- Function "header" syntax:

```
def function_name(parameter_list) :
```

- Use the normal variable naming rules for *`function_name`*
- *`parameter_list`* provides a mechanism for getting values into your function
  - But it's optional
- The **`return`** keyword can be used to send a value out of a function
  - More on this in a bit …

Slides courtesy of Dr. Alan McLeod

The `main()` Function

- Define and call a `main()` function to run your program
  - Convention in Python and many other languages
    - Named `main()` for "mainline logic"
- Use `main()`to call and coordinate other functions
  - Pass data back and forth between them
- Make sure to invoke `main()` to start your program!

Slides courtesy of Dr. Alan McLeod

A Function with Parameters

Here is a (useless) function that displays the sum of two numbers:

```
def addNumbers(num1, num2) :
  sum = num1 + num2
  print("The sum is", sum)
```


Slides courtesy of Dr. Alan McLeod

A Function with Parameters - Cont.

- When you invoke this (useless) function, you need to supply two things for the parameters
  - You supply two numbers as arguments

```
addNumbers(3.4, 6.7)
```

- The code in **`addNumbers()`** runs and the sum displayed
- Within **`addNumbers()`**
  - **`num1`** has the value **`3.4`**
  - **`num2`** has the value **`6.7`**

Slides courtesy of Dr. Alan McLeod

A Function with Parameters - Cont.

- To put it another way …
- The positional arguments **`3.4`** and **`6.7`** have been *mapped* into the parameters **`num1`** and **`num2`**
- **`num1`** and **`num2`** are variables that have been created in the function's parameter list and are *local* to the function

Slides courtesy of Dr. Alan McLeod

Function Returns

- A function may *return* something
  - The "something" can be any Python type
    - A `str`, an `int`, a `float`, *etc.*
- Functions that don't return anything are sometimes called *procedures*
  - Like **`print()`**, for example
- Can you think of some functions that return something?
  - **`input()`**
  - **`float()`**
  - **`str()`**
  - …

Slides courtesy of Dr. Alan McLeod

A Function with a Return Value

- How can **`addNumbers()`** be changed to return the sum instead of printing it out?
  - It is usually regarded as "tacky" to have functions print things instead of returning them
  - Let `main()`do the printing!
    - … except in Assignment 1

```
def addNumbers(num1, num2) :
  sum = num1 + num2
  return sum
```


Slides courtesy of Dr. Alan McLeod

Returning Values

- If you don't have a **`return`** statement, then your function does not return anything
  - It is invoked without expecting any value to come out of the function
    - No assignment required when invoking
- Execution of a function stops as soon as you execute the **`return`** statement

Slides courtesy of Dr. Alan McLeod

The Advantages of Functions

- Each function is a building block for your program

- Construction, testing and design is easier
- Functions avoid code duplication
- Functions make re-use of your code more likely
- Well-written functions reduce the need for extensive comments

Slides courtesy of Dr. Alan McLeod

Designing a Function

- A function should only do one thing
  - If you describe the function and need to use the word "and", then it is probably doing more than one thing
- Try to keep the parameter list as short as possible
  - Later: take advantage of default arguments
- The function itself should be short
  - In the range of 1 to 15 lines, ideally
  - Not larger than can be displayed on the screen
- Functions can be declared inside other functions
  - Known as *nested* functions
  - Avoid unless you have a good reason!

Slides courtesy of Dr. Alan McLeod

Designing a Function - Cont.

- Try to get your function to return something rather than print something
  - Trust your console I/O to a function like `main()`
    - Ignore this suggestion for Assignment 1
- We will discuss some additional topics later that will make your functions easier to write and use
  - Default arguments
  - Keyword parameters
  - Raising exceptions
  - Checking argument types

Slides courtesy of Dr. Alan McLeod

Designing a Function - Cont.

- Choose good, descriptive function and parameter names
  - It should be obvious what the function is doing
- If you only need to add a bit more code to make your function more universally applicable – do it!
- Be prepared to re-structure a working program to get a better design
- By convention, `main()`should always be the starting point of your program

Slides courtesy of Dr. Alan McLeod

Variable Scope

- A variable created inside a function is known inside that function
  - These variables are called *local variables*
- A variable created at the same level as the function headers is known everywhere in the program
  - These variables are called *global variables*
- What do I mean by "known"?

Slides courtesy of Dr. Alan McLeod

Variable Scope – Cont.

- A variable's *scope* is the part of the program where its value can be used
  - Local variables: inside its function
    - And any other functions or statements nested in that function
  - Global variables: everywhere
- Changing the value for a global variable in a function requires an extra step
  - "Re-declare" it using the `global` keyword
- Also watch changing the value of a local variable inside a nested function …

Slides courtesy of Dr. Alan McLeod

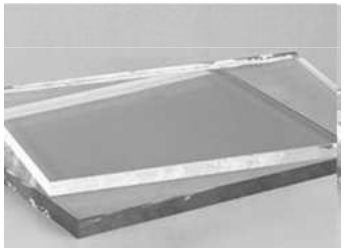
Global Variables

- The problem with globals is that any function can mess with them
  - It is easy to lose track of how they are being used
- Global variables violates the principle of functional isolation!
- Two simple rules
  - Don't declare global variables unless the vast majority of your functions will use this variable
    - You must think your code will be significantly easier to work with and read as a result
  - You can declare constants as global variables
    - The constant's variable name should be in all uppercase

Slides courtesy of Dr. Alan McLeod

Global Constants - Demo

- WindowWeight.py
  - Calculates the weight of a piece of window glass given its dimensions



Slides courtesy of Dr. Alan McLeod

Aside - "Magic Numbers"

- These are numeric literals that just appear in your program
- Sometimes they make sense
  - Like assigning a temporary value to a new variable

```
sum = 0
```

- But sometimes they don't:

```
distance = measurement / 2.54
```

  - Where did **`2.54`** come from and what does it mean?
- Something like this is better:

```
distance = measurement / CM_PER_INCH
```


Slides courtesy of Dr. Alan McLeod

Variable and Function Names

- Follow Python restrictions on names:
  - Use only letters, numeric digits (0 to 9) and the "_" character
  - Cannot start names with a number
  - **Python is case sensitive!**
- Variables and function names usually start with a lower case character
- Constants are all in upper case
- The use of one or two underscores and the beginning and/or the end of a variable name has a special meaning in Python …
- Variable names are usually nouns
- Function names are usually verbs or verbs and nouns

Slides courtesy of Dr. Alan McLeod

Variable and Function Names - Cont.

- Be descriptive, but not excessive!
- Examples:
  - **`numStudents`**
  - **`setPassingGrade(`***`parameter_list`* **`)`**
- Somewhat too long:
  - **`flagThatIsSetToTrueIfAProblemArises WhenThereIsAFullMoonOverMyHouseInTh eWinterWhileMyProgramIsRunning`**

Slides courtesy of Dr. Alan McLeod

Variable and Function Names - Cont.

- Use camelCase for variable names
  - *Google Python Style Guide* says use underscores
  - I don't like that style in Python, personally
- Note that Python keywords are in all lower case
- You will get an error message if you attempt to use a keyword as a variable name
- It is very tacky to use a keyword as a variable name just by changing the capitalization!

Slides courtesy of Dr. Alan McLeod

Spacing

- Use 4 spaces for indentation
- Don't mix tabs and spaces
  - Not a problem if you are only using IDLE
    - When you hit the <tab> key you automatically get 4 spaces
- Long lines:
  - Keep lines < 80 characters in length
  - Use the Python continuation character \
    - Indent a continued line so that it lines up nicely

Slides courtesy of Dr. Alan McLeod

Spacing - Cont.

- Use one blank line above a **`def`** statement
  - No blank lines below
- A blank line inside a function can be used to delineate a block of code
  - Don't put too many blank lines inside a function
  - Don't double space your code!

Slides courtesy of Dr. Alan McLeod

Comments

- When the name of a variable is not self-explanatory, add an inline comment when it is first initialized
- Add comments at the start of logical blocks
  - Indent comment same as start of block
- You don't need to explain code that is obvious
  - Focus on code that is tricky to understand
    - *Maybe it needs to be re-written?*
- # **`TODO`** comments can be used to mark where more work is needed

Slides courtesy of Dr. Alan McLeod