

# Python Programming Environment for Educational Assessment

By

**Christopher Gillum**

A senior thesis in partial fulfillment of  
the requirements for the degree of

**Bachelor of Science  
With Departmental Honors**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY OF WASHINGTON

JUNE 2006

Presentation of work given on May 30, 2006

Thesis and presentation approved by \_\_\_\_\_  
Steven Tanimoto, Project Advisor

Date \_\_\_\_\_

## Contents

<b>1. Introduction</b> .....	2
1.1. Motivation .....	2
1.2. The INFACT Online Learning Environment .....	2
1.3. The GMA Gnome .....	3
1.4. The Integrated Python Development Environment .....	4
<b>2. Python and IDLE</b> .....	4
2.1. The Python Programming Language .....	4
2.2. The IDLE Integrated Development Environment .....	5
<b>3. Integrating with INFACT</b> .....	5
3.1. Design Goals .....	5
3.2. The Python INFACT Client Module .....	6
3.3. The INFACT Web Interface .....	7
3.4. IDLE Modifications .....	8
3.5. Running the Modified IDLE Interpreter .....	12
<b>4. Future Work</b> .....	13
4.1. Use Studies for Validation .....	13
4.2. Propositions for IDLE .....	14
4.3. Propositions for INFACT .....	15

# 1. Introduction

---

## 1.1. Motivation

One problem with traditional methods of teaching introductory programming courses in universities is that it's difficult for an instructor to assess the kinds of understanding problems students often face. Learning is a process, and throughout that process there can be many hurdles that students need to overcome in order to successfully complete a task. Students are not prepared for such hurdles because they are often not discussed or are too obscure to cover in a lecture setting.

This problem is difficult to address over the course of a class because there is often very limited student-teacher interaction. Communication is often limited to office hours or email conversations and this is frequently not sufficient to address or diagnose all the issues face when working on their assignments. These problems are especially prevalent in larger lecture settings where some 100+ students are in enrolled. The use of teaching assistants helps but is not always a viable option because interaction between students and their teaching assistants can suffer from the same set of issues.

Another problem with the traditional model is that it can be difficult to collect and organize responses provided by students who are having problems with their assignments - if the students are comfortable enough to send clear and honest responses to the instructor at all! The little feedback that instructors do receive is typically in the form of verbal communication or scattered emails, and often only represents a small subset of the students.

## 1.2. The INFACT Online Learning Environment

### *INFACT: The context of the Research*

An online learning environment named INFACT, developed with the direction of Steven Tanimoto in collaboration with other departments at the University of Washington, aims to address many of the issues explained previously. An "Online Learning Environment" (OLE) is a networked computing software infrastructure in which software tools are provided to students and teachers to improve the overall quality of learning. INFACT, which is an acronym for **I**ntegrated **N**etworked **F**acet-based **A**ssessment **C**apture **T**ool, is such an environment, and aims to facilitate the use of formative assessment in the teaching/learning process.

### *Formative and Summative Assessment Methods*

Formative assessment can be thought of as a diagnostic evaluation of a student's progress as he or she learns in order to provide constructive feedback and thus to assist the student in the learning process. Formative assessment methods contrast with those of summative

assessment, in which the primary concern is a single score or grade. Furthermore, summative assessment, unlike formative assessment, makes no attempt to direct the student through the process of a course or assignment.

### *Increased Teacher Productivity*

With these challenges in mind, another goal of the INFACT system is to maximize teacher productivity. By helping to provide students with feedback as they work on an assignment using the system, INFACT can assist teachers in identifying learning problems early; and if this diagnosis and feedback leads to a solution, then the student no longer needs to use the teacher's time to get a solution for their problem. These instructors are then free to focus on other issues related to the course.

### *Student Data Aggregation*

A system that is able to diagnose problems that students face is also able to record these problems. Using INFACT and the educational tools built around it, it then becomes possible to collect information about the conceptual problems that all students in a course are facing and have that information analyzed. Having this kind of accessible data then allows the instructor (or some other tool) to look at the data and find common conceptions and misconceptions which then gives the instructor an opportunity to adjust the course in a way to correct the most common misconceptions that are present throughout the class.

## 1.3. The Graphical Model Assessment (GMA) Gnome

The current set of applications designed to work with INFACT, such as the programming environment IDLE (to be described later), collect information about how the students use the application. This information could be as simple as where a student is clicking on a form or could be more detailed such as error messages or the result of a computation. This information is then captured by the INFACT system where it can be automatically assessed. This is where the GMA Gnome, designed by Nathan Evans, comes into play.

A “gnome” is an autonomous software agent that runs on the INFACT server. The GMA gnome is a particular type of gnome that is responsible for diagnosing student conceptions and misconceptions based on data stored about a student within INFACT. The acronym GMA stands for **G**raphical **M**odel **A**ssessment and the gnome was given this name because it performs assessment on student data based on graphical Bayesian models created by an instructor. The GMA gnome runs and analyzes student data in real time and is capable of providing students with feedback in the form of email, dynamically created web content, instant messages, and a variety of other means.

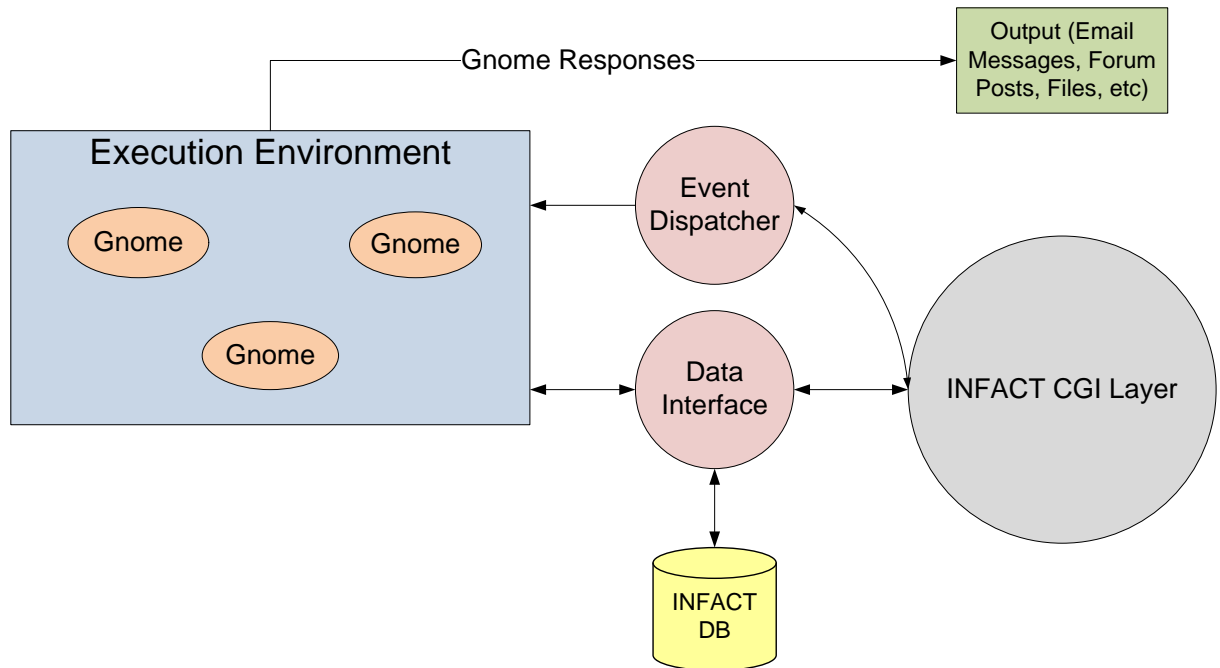


Figure 1: Architecture of INFAC T

The diagram show in Figure 1 points out what the different pieces of INFAC T are and how they interact with each other. Multiple gnomes interact within the Execution Environment, receiving from and querying for information from the event dispatcher and data interfaces components. Gnome output consists of files, messages, web content and other things that are generally intended to be seen by either the student or the instructor.

#### 1.4. The Integrated Python Development Environment

The rest of this paper is going to discuss the core of my project, which was to integrate a programming environment into the INFAC T system to take advantage of INFAC T's assessment tools. The goal is to create an environment where students can learn to write simple expressions and algorithms in the Python programming language. To accomplish this, an existing Integrated Development Environment (IDE) called IDLE, developed by the open source community responsible for the Python programming language, was taken and modified. We will dive more into the details of this integration work after a brief introduction to Python and IDLE in the following section.

## 2. Python and IDLE

---

### 2.1. The Python Programming Language

Python is an interpreted programming language created by Guido van Rossum in 1990. Python is fully dynamically typed and uses automatic memory management; it is thus similar to Perl, Ruby, Scheme, Smalltalk, and Tcl. Python is developed as an open source project and is managed by the non-profit Python Software Foundation ("Wikipedia", 2006). Python was chosen for the target programming language of this project because it is a good language with which beginning programming students can learn general programming practices without much overhead (e.g. the overhead of learning platform-dependent features such as pointers or the philosophies of object-oriented programming). Python is also available on a wide variety of platforms and thus does not restrict students to any particular platform.

### 2.2. The IDLE Integrated Development Environment

IDLE is the name of the integrated development environment that was created by the open source community that has been maintaining the Python language. It is written entirely in Python and has been modified to integrate with the INFACT system for the purposes of this research project. IDLE was chosen for this project because its source code was available and required no compilation (Python is a scripting language). The requirements for running IDLE are an up-to-date installation of Python (2.4.3 at the time of this writing) with Tkinter (a graphical user interface toolkit) support installed (Lundh, 1999).

It should be noted, however, that the IDLE project itself has no direct relationship with INFACT or the research behind INFACT. IDLE is an IDE designed specifically for writing Python code but is freely available along with its source code from [www.python.org](http://www.python.org) and is distributed under the terms and conditions of the Python 2.4.3 License. One can download and view a copy of this license at <http://www.python.org/download/releases/2.4.3/license/>.

The primary interface of IDLE consists of a single window that imitates a simple interactive command shell with syntax highlighting. This means that it allows users to type Python code one statement at a time and see the results immediately. This works very well for students who are learning a programming language for the first time because feedback is nearly instantaneous. There are also more advanced features such as a debugger, a class browser, and a mode for writing Python code in a non-interactive mode, but these features are not important for the purposes of this project. With this interface students can write single-line statements, loops, function definitions, class declarations, import modules, and anything else that a normal Python script allows.

# 3. Integrating with INFACCT

---

## 3.1. Design Goals

One of the goals of this project was to create a reusable module that would encapsulate the routines required to communicate with INFACCT. In essence, a client-side API that could be used by other Python-based applications that wished to communicate with INFACCT. Additionally, it was intended that the existing IDLE code should be modified as lightly as possible. One reason for this is because Python and IDLE are active open-source projects that are constantly being updated with newer versions. A modification that makes controlled changes to the existing code base can more easily be ported to later releases of the IDLE software.

The modified IDLE code is also designed to run on multiple platforms. The additional code was written specifically to be run on Windows NT and Unix-based systems such as Windows 2000/XP, Linux, and FreeBSD. The application has also been tested and is functional on Windows Vista (Beta 2). Support for Macintosh platforms such as Mac OSX have not been tested but should work in theory since Mac OSX supports the Python language.

## 3.2. The Python INFACCT Client Module

The module that was created for INFACCT integration is written as a single Python script and contains functionality for performing INFACCT authentication, database selection, session management, and event logging both to disk and to INFACCT via HTTP calls to CGI scripts that live on the INFACCT web server. It can be found with the modified IDLE source distribution at `infact-idle/src/infact.py`. This python module is intended to be used by any Python-based application that needs to communicate with the INFACCT system.

### *INFACCT Login*

The INFACCT client module also contains code which displays a graphical user interface for interactive INFACCT login. To use this graphical login, the running system must have the Tkinter Python libraries configured for its Python installation. This toolkit is not included in the modified IDLE source code.

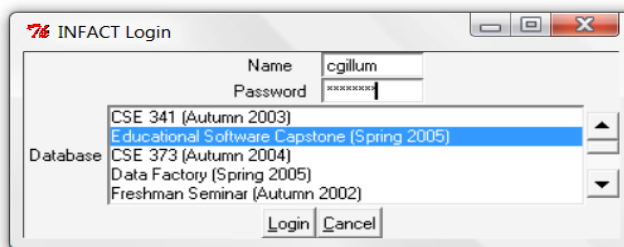


Figure 2: INFACCT client login on Windows Vista

The graphical login utility allows a user to enter a username, hidden password, and select an INFACT database (corresponding to a course registered within the INFACT system). These course databases are determined at runtime by the INFACT module making calls to the INFACT service over HTTP.

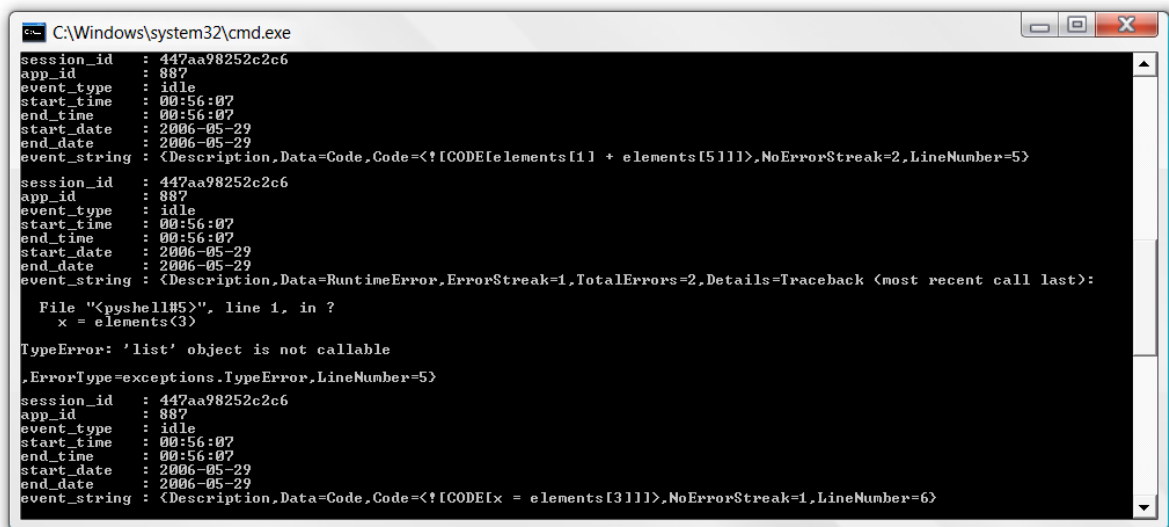
The INFACT login process is completely independent of the INFACT application being used. This means that applications other than the IDLE-based IDE can also use this utility to log into the INFACT system. If login is successful, INFACT returns a session ID back to the client in the form of a string that resembles a GUID. This session ID is maintained by the INFACT client module and used to identify the authenticated client for all subsequent communication with the INFACT system.

### *Event Logging*

The INFACT client module supports two types of logging: Logging to a file on disk and logging to INFACT. Each log entry created by the INFACT module contains the following information:

- **Application ID:** An identifier that identifies the application that is logging information
- **Session ID:** An identifier that identifies the login session of the user
- **Course Database:** The name of the database that corresponds to the class being taught
- **Event String:** Event data in text form
- **Start/End Date/Times:** Timestamp information about when the event data was collected

The most important information comes from the Event String, which is typically a formatted list of flags and/or key-value pairs with information about actions a student is taking in the context of the application being used (e.g. clicking certain icons, making calculations, etc). This information is formatted and then logged to disk and to INFACT.



```
C:\Windows\system32\cmd.exe
session_id : 447aa98252c2c6
app_id : 887
event_type : idle
start_time : 00:56:07
end_time : 00:56:07
start_date : 2006-05-29
end_date : 2006-05-29
event_string : <Description,Data=Code,Code=<![CODE|elements [1] + elements [5 ]]|>,NoErrorStreak=2,LineNumber=5>

session_id : 447aa98252c2c6
app_id : 887
event_type : idle
start_time : 00:56:07
end_time : 00:56:07
start_date : 2006-05-29
end_date : 2006-05-29
event_string : <Description,Data=RuntimeError.ErrorStreak=1,TotalErrors=2,Details=Traceback (most recent call last):

File "<pyshell#5>", line 1, in ?
x = elements(3)

TypeError: 'list' object is not callable

,ErrorType=exceptions.TypeError,LineNumber=5>

session_id : 447aa98252c2c6
app_id : 887
event_type : idle
start_time : 00:56:07
end_time : 00:56:07
start_date : 2006-05-29
end_date : 2006-05-29
event_string : <Description,Data=Code,Code=<![CODE|x = elements [3 ]]|>,NoErrorStreak=1,LineNumber=6>
```

Figure 3: A console windows displaying the contents of a log file



### 3.3. The INFACT Web Interface

#### *Course Administration*

Before any integration with INFACT can be performed, it is required that the course database be created in the INFACT system. Additionally, users (administrators, instructors, and students) must be configured with login names and passwords. This can all be done by an existing administrator from the main INFACT website. This website also allows instructors to create and configure student forums, create graphical assessment models (which are utilized by the GMA gnome) and perform a variety of other administrative tasks via an assortment of web-based tools.

#### *Communication Protocols*

There are at least two methods for enabling an application to communicate with INFACT. The first is through the web via CGI scripts that can be invoked using standard HTTP. The other method makes use of Java's Remote Method Invocation (RMI) protocol. For the INFACT client module, the direct HTTP approach was selected because Python libraries exist that support HTTP communication in addition to the fact that the CGI scripts on the INFACT server accept text data as inputs. RMI typically requires Java objects, which creates a difficulty for non-Java-based applications such as the Python-based IDLE.

Another advantage to the HTTP approach is the ease of testing and debugging. To perform operations, all that is needed is a web browser and a long URL query string. INFACT would then display the result of the operation there on the browser. This was crucial in the integration process when originally creating the INFACT module. For future projects, this HTTP approach is highly recommended.

#### *Authentication*

The INFACT web interface is also responsible for maintaining and enforcing authentication. The client application must provide a session token generated as a result of a successful INFACT login in order to perform non-trivial operations on INFACT. It should be noted that these session tokens expire after a specified amount of time to help enforce security. The session token provided is a 64-bit integer in hexadecimal form (example: 42e810410b77ec (the first two zeros are omitted)).

### 3.4. IDLE Modifications

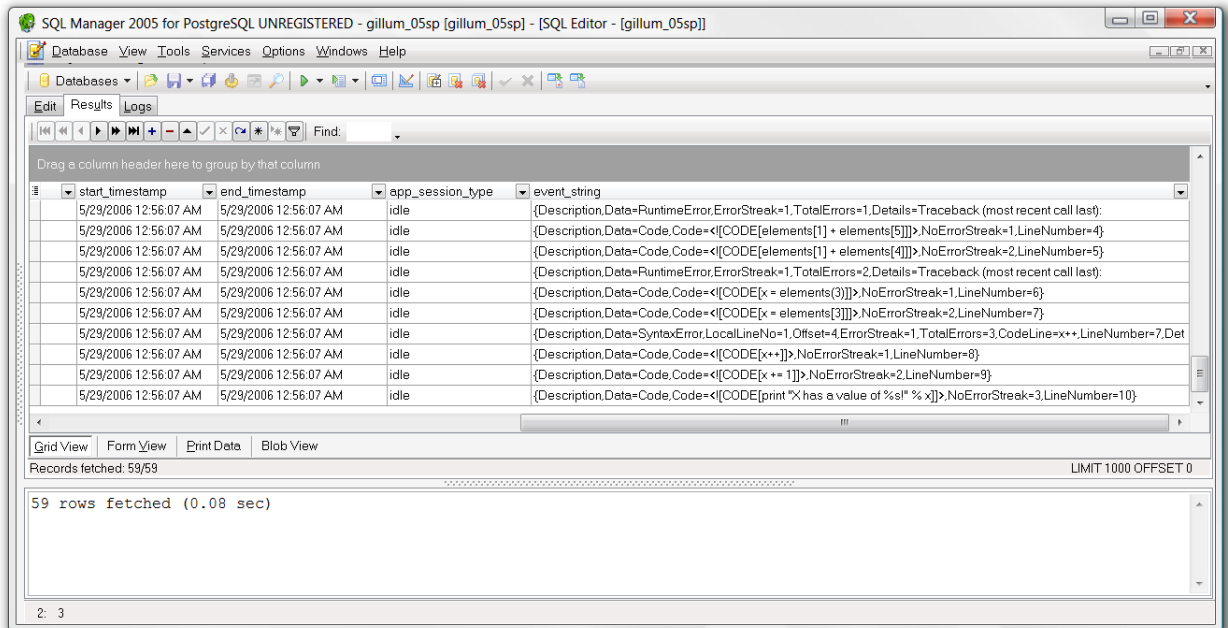
As was discussed in the design goals section, it was intended that the modifications made to the existing IDLE code base be as minimal as possible for the purposes of maintainability. This section will go into more detail about the work that was actually done as well as present the ideas behind the decisions that were made.

## File Structure of IDLE

As was stated previously, IDLE is composed entirely of Python scripts, all of which are located in the `infact-idle/src` directory. There are several scripts in this directory, but the only one that was modified was the `PyShell.py` Python script. This module contains definitions for the high-level IDLE interpreter and is essentially all that was needed to log the most useful information since the student interacts directly with the interpreter UI.

## Information Logged

The core of taking advantage of the INFACT system and the GMA assessment capabilities is logging information related to what the student is working on. It is this information that GMA models will use to determine what conceptions or misconceptions students are faced with. In the modified IDLE IDE, all logging is triggered by submitting Python code to the interpreter. The submitted code is logged as well as any errors that were generated by executing the code.



**Figure 4: A database application displaying what captured data looks like in the INFACT database. Note that this particular application has no relation to INFACT, IDLE, or any other project related to this research.**

It should be noted, however, that a decision was made not to log the output of any computation. Logging program output to INFACT would certainly be something that instructors would find useful. It potentially could make it easier to verify program correctness (or lack thereof) but it also introduces an array of problems. For example, if a student writes a piece of code that goes into an infinite loop while generating output in each iteration, it is possible that the student could inadvertently bring down the INFACT system by flooding it with data and

congesting the network. It's possible to add checks for this situation in the IDLE code, however, and this could possibly make a great future addition to the modified IDLE interpreter.

### Capturing Source Code

Source code at the statement level provides an excellent means of analyzing whether or not a student understands the tasks that they are assigned. It presents a balance between logging too much information, such as an entire source file, and too little information, such as individual expressions or variable names. Because IDLE runs as an interactive interpreter that accepting one statement at a time, this is the natural approach for source code logging.

In IDLE, when source code is submitted from the prompt, it is sent to the Python execution engine which interprets and executes the code. To log the source code statements, a man-in-the-middle approach was used to intercept the code, package it into a format understood by INFACT, and serialize it across the web to the INFACT logging service.

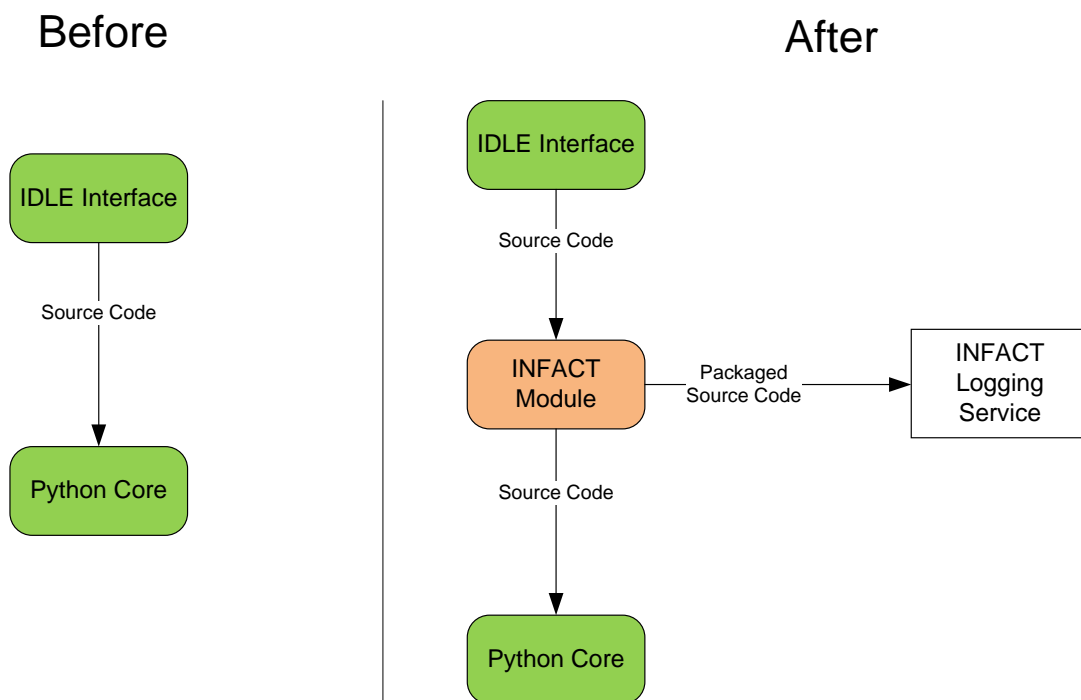


Figure 5: A diagram displaying how IDLE handles source code before and after being modified to work with INFACT

This implementation is simple and straightforward and accomplishes the goal of simplicity in regards to not greatly disturbing the existing IDLE code. In Figure 5, the green nodes represent existing components of IDLE, the red represents the INFACT Python module added, and the white node represents the HTTP interface to the INFACT logging service.

## Capturing Error Information

Source code alone, while valuable, is certainly not sufficient for accurate assessment of student conceptions and misconceptions. Error information is just as important, if not more important, for determining what concepts students are struggling with, especially at the early stages of learning a new programming language.

Two general types of errors were considered in this project: syntax errors and runtime errors. Syntax errors typically indicate some kind of misunderstanding with how to use the language in a general, high-level sense. Students new to a programming language are especially prone to making syntax errors. In essence, they are not necessarily indicative of a problem in determining how to solve a problem, but rather indicative of a problem in determining how to use a language like Python to solve that problem.

Runtime errors as a whole, on the other hand, are not as indicative of any particular misconception, yet typically provide evidence that a student is struggling in their attempt on figuring out how to solve a problem. Looking for specific types of runtime errors (e.g. Python's `IndexError`) provides important clues about the nature of a student's problems. For example, the prevalence of the `IndexError` in a student's log trace combined with other contextual information might provide evidence that a student is struggling with loops and accessing the indexes of an array within that loop. For this reason, the type of a runtime error in addition to the associated call stack is logged to INFACCT.

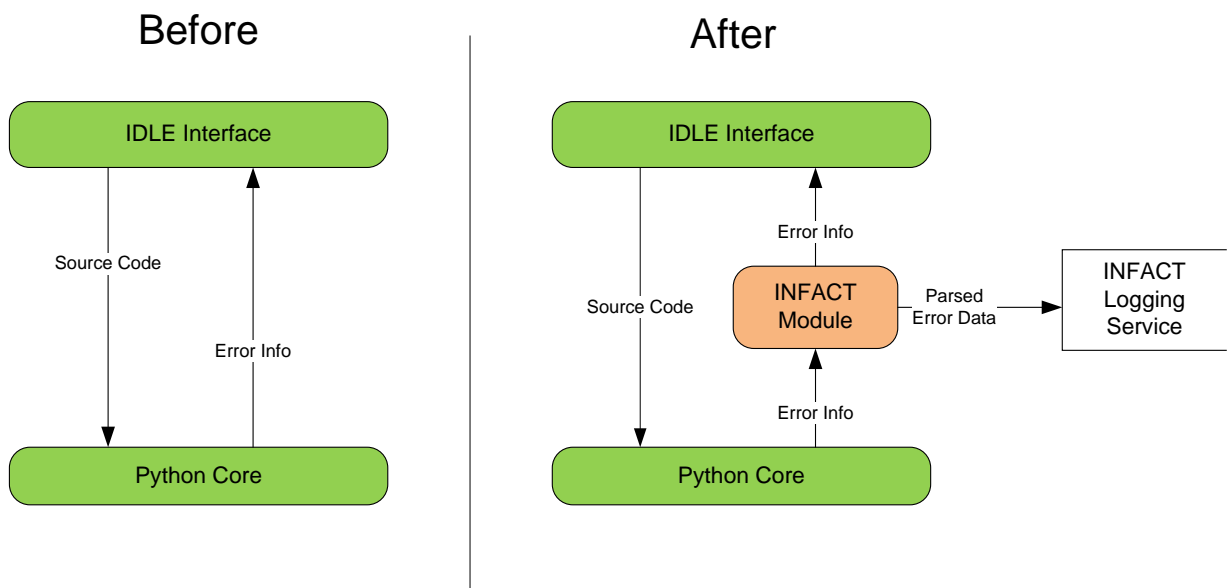


Figure 6: A diagram displaying the changes made to IDLE to handle error logging. Note that the modification for source logging as show in Figure 5 was left out of this diagram for brevity.

As indicated in Figure 6, error logging works very similarly to source logging in the modified IDLE IDE. In the original implementation (without the INFACT additions), source code was sent to the Python Execution Engine (or “Python Core” in the diagram) and error information was sent back up to the IDLE interface. The man-in-the-middle approach is once again used to capture and log this useful error information. When the error information is on its way back to the IDLE interface, it is intercepted by the INFACT module, parsed for useful information (call stack, line number, etc), serialized and sent to the INFACT logging service over HTTP. The original error information is then forwarded to the IDLE interface where it is displayed to the student.

### *Miscellaneous Information that is Logged*

In addition to source code, syntax errors, and runtime errors, the modified IDLE interpreter keeps track of miscellaneous information such as the number of successfully executed statements (and how many were consecutively executed) as well as the total number of errors encountered (also keeping track of error-streaks in the code entry). This information complements the rest of the logging by providing more summative data to INFACT. For example, a large number of consecutive errors might indicate that a student is guessing or is having an unusually difficult time trying to debug a problem.

## 3.5. Running the Modified IDLE Interpreter

### *Startup Requirements*

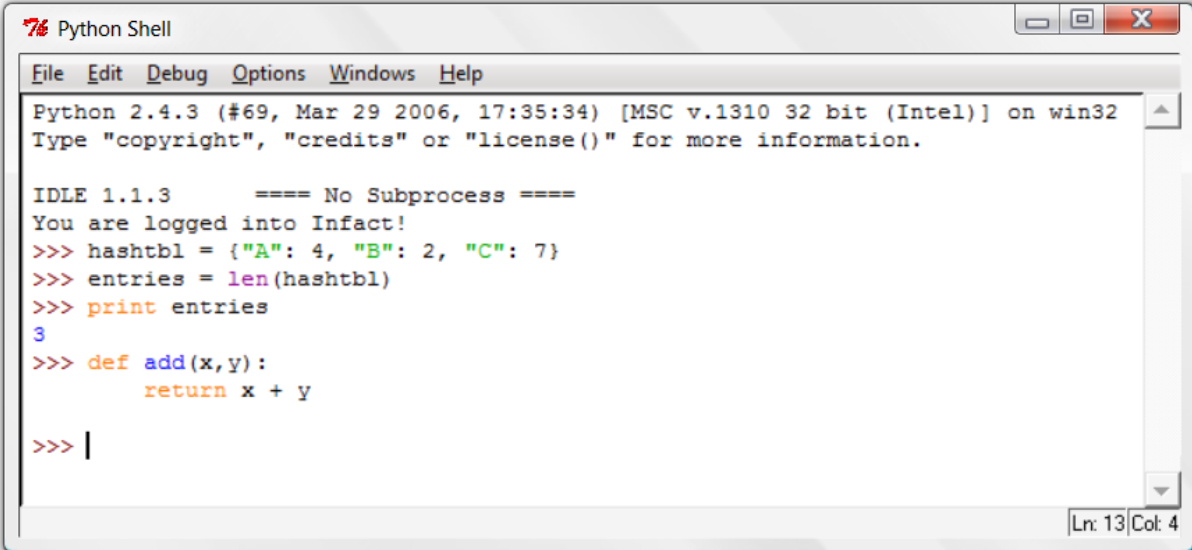
To start the modified IDLE application, the user must execute PyShell.py with a “-n” flag passed as the first argument. This “-n” flag changes the behavior of the IDLE interpreter to manage the execution of code within its own process. Normally, IDLE forks a new Python instance and forwards all requests for code execution to this new child process. This presents a major problem to the current error-logging implementation that was written for IDLE because error information becomes exposed in the child process and not in the IDLE process. This would then require the modification of Python itself to capture the error information! The “-n” flag prevents this subprocess behavior as indicated by the “No Subprocess” message in the screenshot of Figure 7.

### *The Login Process*

Once the user has started the modified IDLE application, they are presented with the login interface of the INFACT module shown in Figure 2. If the student logs into INFACT successfully, the message “You are logged into Infact!” appears on the interpreter and the student can immediately start writing Python code. If login fails, the student may try to login again as many times as needed. If the student clicks “Cancel”, this will abort the login process. In the event of an aborted login, the student may still use the modified IDLE to write Python code, but all logging functionality will be disabled.

## Interacting with the interpreter

After the login process is complete, the student is presented with a prompt where he or she may begin typing Python statements. Each ">>>" in the interpreter indicates the start of a new statement. If a statement is an expression or writes output to standard out, the expression is evaluated and the result is displayed in blue below the prompt. If the statement is not an expression (e.g. a function definition or assignment), no feedback is given. In the case that one of the statements or expressions results in an error, the error information is displayed below the prompt in red. If it is a syntax error, the character that violated the syntax rules of Python is highlighted in red as well. In all cases, the code written by the user is submitted to INFAC T without notifying or interrupting the student, allowing them to continue their work and write great Python code.

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Debug", "Options", "Windows", and "Help". The main text area shows the following content: "Python 2.4.3 (#69, Mar 29 2006, 17:35:34) [MSC v.1310 32 bit (Intel)] on win32", "Type 'copyright', 'credits' or 'license()' for more information.", "IDLE 1.1.3", "==== No Subprocess ====", "You are logged into Infact!", ">>> hashtbl = {'A': 4, 'B': 2, 'C': 7}", ">>> entries = len(hashtbl)", ">>> print entries", "3", ">>> def add(x,y):", " return x + y", and ">>> |". The status bar at the bottom right shows "Ln: 13|Col: 4".

```
Python 2.4.3 (#69, Mar 29 2006, 17:35:34) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

IDLE 1.1.3      ==== No Subprocess ====
You are logged into Infact!
>>> hashtbl = {'A': 4, 'B': 2, 'C': 7}
>>> entries = len(hashtbl)
>>> print entries
3
>>> def add(x,y):
    return x + y

>>> |
```

Figure 7: A screenshot of the modified IDLE in action

To exit the interpreter, the student must choose "Exit" from the File drop-down menu or press [Ctrl]+D on the keyboard (a NULL terminator in the Unix world). Exiting the application terminates the user's INFAC T session and the user must login once again and begin a new session the next time the interpreter is loaded.

# 4. Future Work

## 4.1. Use Studies for Validation

One of the most important tasks of any research project is to go out into the field and put to test the written hypothesis and constructed works. This project is no exception. The next step is to find out what can actually be learned by assessing facet-based data from students who using this Python Programming Environment.

To do this, students will be needed who are interested in learning Python and who aren't already masters of programming in general. A series of assignments can be used to teach certain aspects of Python programming, such as the concepts of lists, conditionals, and loops. For each of these assignments, an instructor will need to construct a graphical model for use with the GMA Gnome component of INFACT. This can be done with the existing graphical tools found on the INFACT website.

Figure 8 shows an example of a graphical model that could potentially be used to assess whether students understand how to use Python loops to solve a certain type of iteration problem. This particular model allows for a successful completion of a for-loop or a certain number of syntax errors before sending feedback indicating success or failure. For more information about the components and development criteria of graphical models within the GMA subsystem, refer to the paper by Nathan Evans titled "[The INFACT GMA Gnome: Providing Automated Student Feedback via Graphical Models](#)".

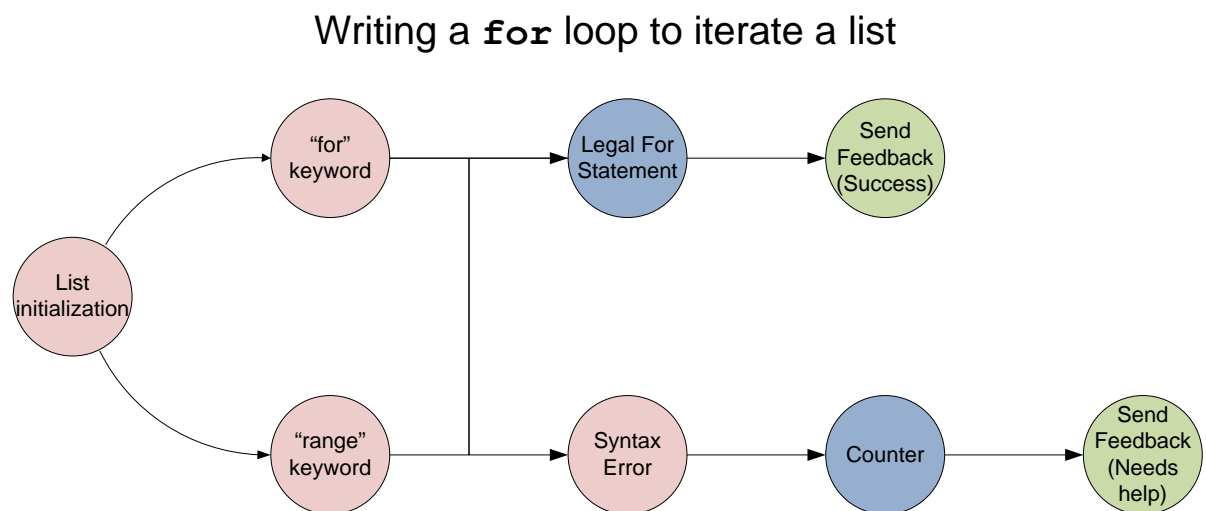


Figure 8: An example of a graphical model for assessing whether students understand the for-loop syntax in Python

## 4.2. Propositions for IDLE

### *Logging of Additional Metadata*

What is logged is a big part of how effective INFACT assessment can be for any particular application. Currently, the modified IDLE interpreter logs source code, error information and statistical error information. Future versions of this interpreter could go beyond that, however, and perform more detailed tracking. For example, when IDLE is run in non-interactive mode (the UI is a blank free-form editor), extensions could be added that calculate the number of class definitions, local variables, track inheritance trees etc. It would take more work to implement, but this type of logging could open up new doors in the assessment of students learning Python and more complicated features such as object oriented programming.

### *Batch Logging*

One potential issue with the current implementation of this IDLE-based interpreter is that source code is logged immediately to INFACT. In cases where bandwidth is scarce, this could cause network congestion when many students are concurrently using the application. A potential solution to this would be to have the interpreter cache log entries in memory and send batches of event updates to INFACT on a periodical basis.

### *Instant Integrated Feedback Window*

Currently, a student must periodically check the forums or their email accounts in order to see any feedback from the INFACT system while using the modified IDLE IDE. A better solution that allows students to stay within the IDLE context would be to add a window to the IDE where all textual feedback is automatically and instantly delivered. The INFACT project currently has a similar instant feedback application known as IWindow, but this application runs as a separate process. An integrated window would be an ideal add-on to the IDLE IDE.

## 4.3. Propositions for INFACT

### *Moving on From Perl*

Integrating the IDLE application with the INFACT system was no easy task. This was in part due to the fact that the CGI scripts that drive the web interface are all written in Perl. Perl is a powerful language, but falls short in terms of readability and maintainability, especially when the original developer is not available to help debug integration issues. From that standpoint, a good move would be to rewrite these interfaces in a more enterprise-ready platform such as Java or .NET/Mono. Not only do these platforms scale to the enterprise-level (which is essential for INFACT to be scaled to large deployments) but they also have native support for XML web services and other features that could greatly ease the burden of integration and simplify the task of maintaining INFACT.



### *INFACT API Libraries*

As was implied in the last section, integration was a difficult task in part because it required thorough analysis of the source code (which was written in Perl). Part of the work in this project was to create a module that could be used by other Python-based applications to more easily integrate with INFACT. A longer-term solution would be to develop a common set of client libraries that could be easily imported into any application that wanted to integrate with INFACT. This library would act as the interface between all INFACT applications and the INFACT web front-end, allowing for greater maintainability of both INFACT and its client applications by having this common library control all access to INFACT.

### *Diagnostic Tools*

The last item in the maintainability and integration category would be diagnostic tools for INFACT. Currently, if information is not properly logged, there is no way to discover why unless the cause for error was logged to the error log file on the INFACT server. Adding features such as instrumentation or perhaps adding debug and error logging features could help developers more quickly solve integration problems when writing applications for the INFACT system. One possible way to do this would be to develop a new type of Gnome that stores and sorts diagnostic information so that it can be made available for INFACT clients.

# Acknowledgements

---

I would like to thank Professor Steven Tanimoto and Nathan Evans for their invaluable assistance and support with this project. Also to be acknowledged is Professor Richard Ladner for hosting my research talk, which was coincidentally the 100<sup>th</sup> undergraduate research seminar hosted for the department of Computer Science and Engineering at the University of Washington.

## Bibliography

*Wikipedia*. (2006, May 30). Retrieved June 1, 2006 from Python Programming Language: [http://en.wikipedia.org/wiki/Python\\_programming\\_language](http://en.wikipedia.org/wiki/Python_programming_language).

Benson, N. J. (2004, June). The INFACT Gnome Environment: A Platform for Autonomous Agents Generating Automatic Student Assessments and Feedback. University of Washington, CSE.

Evans, N. (2005, June). The INFACT GMA Gnome: Providing Automated Student Feedback via Graphical Models. University of Washington, CSE.

Lundh, F. (1999). *An Introduction to Tkinter*. Retrieved from <http://www.pythonware.com/library/tkinter/introduction/>.

---