# ISAT 252 Spring 2011

“Learning is not compulsory...neither is survival.”

—W. Edwards Deming

## Syllabus

“Computational Thinking” is a term coined by Jeanette Wing, Chair of the Computer Science Department at Carnegie-Mellon University. According to Wing, Computational Thinking:[raw]

> is a way of solving problems, designing systems, and understanding human behavior that draws on concepts fundamental to computer science. To flourish in today’s world, computational thinking has to be a fundamental part of the way people think and understand the world.
>
> means creating and making use of different levels of abstraction, to understand and solve problems more effectively.
>
> means thinking algorithmically and with the ability to apply mathematical concepts such as induction to develop more efficient, fair, and secure solutions.
>
> means understanding the consequences of scale, not only for reasons of efficiency but also for economic and social reasons.

[/raw]
If you think about these points, you’ll realize that ***computational thinking is NOT just for programmers, but represents a general approach to problem solving useful in most all disciplines***. That is why the ISAT 252 course is a required part of the ISAT curriculum. You should begin to find the skills you learn here to be useful in a broad array of the problem-solving activities you’ll encounter in life.

> The most important thing you will learn in this class is NOT how to program, but rather how to approach a problem, analyze a problem, design a solution to a problem, and implement a solution to a problem using strategies and techniques developed and honed in the programming community.

You will also learn how to write real, practical, working programs.

Forums

Announcements

252 Events

*There are no events.*

Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

## Suggested Objectives

Many problems may be solved with computers and there are many skill sets one might acquire in pursuit of this goal. As your instructor, I have two objectives, that you:

1. ***engage seriously with the content***–you (or somebody) spend too much money for you to be here to waste this opportunity
2. ***stretch*** yourself–if it doesn't hurt a little, you're not doing it right

Ultimately, though, your objectives for this class need to be your own. Here are some suggestions of things you might like to come away with, the ability to:

1. Demonstrate basic familiarity with computer hardware functions
2. Recognize when problems in science and business are amenable to software solutions
3. Analyze such problems and generate plans for implementing software solutions including elements such as class diagrams, flow charts, and pseudocode
4. Implement relatively simple software applications with graphical user interfaces
5. Employ procedural programming constructs effectively: variables and constants, conditional expressions, flow control, arrays, modularity with sub-procedures and functions, and input/output with text files
6. Employ event-driven programming constructs effectively: built-in and custom event-handling procedures, catching and handling exceptions
7. Employ simple object-oriented programming constructs: distinguish between objects and classes, identify when classes would be useful in a program, implement a basic class, and use that class in a program
8. Use a range of debugging techniques
9. Document code appropriately to increase ease of maintenance and understanding
10. Read and understand code that others have written with an eye towards code re-use
11. Describe knowledge-based systems and the problem types to which they are best suited
12. Build a simple KBS using appropriate architecture and methodology
13. Use 3 forms of knowledge representation: pseudo code rules, decision trees and rules in a program/shell rule base
14. Extract production rules from a narrative, identify key variables, and construct decision trees in an applied problem-solving scenario
15. Evaluate a given problem within its social context and identify an appropriate paradigm within which to develop a software solution

## Suggested Objectives

Many problems may be solved with computers and there are many skill sets one might acquire in pursuit of this goal. As your instructor, I have two objectives, that you:

1. ***engage seriously with the content***–you (or somebody) spend too much money for you to be here to waste this opportunity
2. ***stretch*** yourself–if it doesn't hurt a little, you're not doing it right

Ultimately, though, your objectives for this class need to be your own. Here are some suggestions of things you might like to come away with, the ability to:

1. Demonstrate basic familiarity with computer hardware functions
2. Recognize when problems in science and business are amenable to software solutions
3. Analyze such problems and generate plans for implementing software solutions including elements such as class diagrams, flow charts, and pseudocode
4. Implement relatively simple software applications with graphical user interfaces
5. Employ procedural programming constructs effectively: variables and constants, conditional expressions, flow control, arrays, modularity with sub-procedures and functions, and input/output with text files
6. Employ event-driven programming constructs effectively: built-in and custom event-handling procedures, catching and handling exceptions
7. Employ simple object-oriented programming constructs: distinguish between objects and classes, identify when classes would be useful in a program, implement a basic class, and use that class in a program
8. Use a range of debugging techniques
9. Document code appropriately to increase ease of maintenance and understanding
10. Read and understand code that others have written with an eye towards code re-use
11. Describe knowledge-based systems and the problem types to which they are best suited
12. Build a simple KBS using appropriate architecture and methodology
13. Use 3 forms of knowledge representation: pseudo code rules, decision trees and rules in a program/shell rule base
14. Extract production rules from a narrative, identify key variables, and construct decision trees in an applied problem-solving scenario
15. Evaluate a given problem within its social context and identify an appropriate paradigm within which to develop a software solution

## Forums

Announcements

## 252 Events

*There are no events.*

## Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

## Suggested Objectives

Many problems may be solved with computers and there are many skill sets one might acquire in pursuit of this goal. As your instructor, I have two objectives, that you:

1. ***engage seriously with the content***–you (or somebody) spend too much money for you to be here to waste this opportunity
2. ***stretch*** yourself–if it doesn't hurt a little, you're not doing it right

Ultimately, though, your objectives for this class need to be your own. Here are some suggestions of things you might like to come away with, the ability to:

1. Demonstrate basic familiarity with computer hardware functions
2. Recognize when problems in science and business are amenable to software solutions
3. Analyze such problems and generate plans for implementing software solutions including elements such as class diagrams, flow charts, and pseudocode
4. Implement relatively simple software applications with graphical user interfaces
5. Employ procedural programming constructs effectively: variables and constants, conditional expressions, flow control, arrays, modularity with sub-procedures and functions, and input/output with text files
6. Employ event-driven programming constructs effectively: built-in and custom event-handling procedures, catching and handling exceptions
7. Employ simple object-oriented programming constructs: distinguish between objects and classes, identify when classes would be useful in a program, implement a basic class, and use that class in a program
8. Use a range of debugging techniques
9. Document code appropriately to increase ease of maintenance and understanding
10. Read and understand code that others have written with an eye towards code re-use
11. Describe knowledge-based systems and the problem types to which they are best suited
12. Build a simple KBS using appropriate architecture and methodology
13. Use 3 forms of knowledge representation: pseudo code rules, decision trees and rules in a program/shell rule base
14. Extract production rules from a narrative, identify key variables, and construct decision trees in an applied problem-solving scenario
15. Evaluate a given problem within its social context and identify an appropriate paradigm within which to develop a software solution

“Marge, I can't wear a pink shirt to work. Everybody wears white shirts. I'm not popular enough to be different.”

—Homer Simpson

## Objective: Develop Your Soft Skills

The Partnership for 21st Century Skills identifies a set of learning and innovation skills they refer to collectively as the “Four C’s”:[raw]

- Creativity and Innovation
- Critical Thinking and Problem Solving
- Communication
- Collaboration

[/raw]
These skills are skills you should be working on in EVERY COURSE YOU TAKE.  Most of the time we don’t put them explicitly on the syllabus.  As your prof, I want you to make these a priority and see that learning to communicate effectively, be creative, and work effectively on teams as every bit as important, if not more so, than the content objectives for the course.

## Forums

Announcements

## 252 Events

*There are no events.*

## Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

## Footer widgets

This is a temporary widgetized area. Please go to **Appearance >> Widgets** to configure it.

Twitter RSS feed Go to top ↑

“One of the marks of having a gift is to have the courage to use it.”

—Katherine Anne Porter, US Novelist

# Objective: Develop Your Creativity

## Creativity means taking risks.

You cannot propose a new, unorthodox approach to solving a problem unless you have the courage to be wrong. As I indicate in the objective for identifying problems:

> There are children playing in the street who could solve some of my top problems in physics because they have modes of sensory perception that I lost long ago.

J. Robert Oppenheimer
One of the physicists responsible for the creation of the atomic bomb
Borrowing from the Partnership for 21st Century Skills website, here are some objectives you may want to adopt in this class:

## Think Creatively:

[raw]

- Use a wide range of idea creation techniques (such as brainstorming)
- Create new and worthwhile ideas (both incremental and radical concepts)
- Elaborate, refine, analyze and evaluate their own ideas in order to improve and maximize creative efforts

[/raw]

## Work Creatively:

[raw]

- Develop, implement and communicate new ideas to others effectively
- Be open and responsive to new and diverse perspectives; incorporate group input and feedback into the work
- Demonstrate originality and inventiveness in work and understand the real world limits to adopting new ideas
- View failure as an opportunity to learn; understand that creativity and innovation is a long-term, cyclical process of small successes and frequent mistakes

[/raw]

## Implement Innovations:

[raw]

- Act on creative ideas to make a tangible and useful contribution to the field in which the innovation will occur

[/raw]

## Who cares? Why do this?

Some people have come to believe that they are not creative.  They think that creativity is one of those things that you are either born with or not.  **This is not true**.  Creativity is a skill, just like anything else, and you can get better at it through practice.  Higher levels of creativity are associated with higher quality problem solving, which ultimately means you'll have more fun at your job and make your employers happier.

## Rubric for Evaluation

How do you know if you are being creative?  How do you judge creativity?  One of the main reasons you don't see "creativity" as part of your grading in your classes is that by definition you can't grade somebody on something that isn't specified, and if it's truly creative, there's no way to specify it ahead of time.  It's a Catch 22 and one of the main reasons that creativity isn't taught more in schools.  In order to evaluate your level of creativity, try asking yourself the following questions.  Ultimately, you'll have to decide for yourself if the answers mean you are developing creativity or not.[raw]

**Have you explored a variety of creativity techniques?**
There are many processes for developing creativity such as assumption busting, boundary examination, brainstorming, and interestingly "do nothing."  Have you researched these techniques?  Have you tried any of them?

**Have you reflected on your frustrations?**
It is often said that necessity is the mother of invention.  Frequently our best ideas come when we are stuck.  But you need to give yourself time and quiet and be intentional about reflecting on them.  Have you done this?

**Have you asked other people what they think?**
Creativity is often a label that is attached after the fact to novel approaches to attacking a problem.  Have you tried asking other people to rate the creativity of your approach?

[/raw]

## Ideas for Learning This Stuff

Hopefully the text above should have given you enough ideas to get started on this.

## Conclusion

Creativity is one of the critical skills you'll need to be successful in your career.  Take time in this class and in your other classes to work on developing this skill.

“...there's a reason education sucks...they don’t want a population of citizens capable of critical thinking.”

—George Carlin

## Objective: Develop Critical Thinking Ability

To be honest, this entire course is a practice in critical thinking. I’ve included this page for completeness because it is one of the Four C’s of 21st Century Skills, but if you are making a concerted effort to accomplish the other objectives in this course, you’ll be making great strides toward improving your critical thinking ability.

Here’s the George Carlin video that’s the source of the quote at the top of this page:

"There are children playing in the street who could solve some of my top problems in physics, because they have modes of sensory perception that I lost long ago."

--J. Robert Oppenheimer

# Objective: Identify Key Problems

## Our Mantra: No software for software's sake.

Nobody builds software for the sake of building software. **Software is always built to solve a problem.** One of the most important steps in any software project is coming up with the idea for what software to build. Here are some suggested learning objectives in the area of proposing software projects. You might like to be able to:
[raw]

- Recognize when problems in science, engineering, business, or society are amenable to software solution
- Identify and describe the social context of problems for which software solutions are proposed
- Evaluate the benefits and costs of implementing the solution at a particular scope, or potentially at multiple scopes if appropriate
- Write a successful proposal to an organization, business, professor, or funding entity (e.g. the NSF) for software you'll build to solve a problem for them

[/raw]

## Who cares? Why do this?

Paul is taking a chemistry class for which he has to write a weekly lab report. Week after week he finds himself cutting and pasting large amounts of data into various parts of an Excel spreadsheet, entering in the same formulas, producing graphs and finally copying the graphs into the Word document he has to turn in. He wishes there was only some way to automate some of these repetitive and boring tasks…

Rachel has a summer internship at a technology consulting company. The company makes their money by writing proposals for clients and building software for them that improve their business processes. Rachel finds that she's really good at identifying what problems companies tend to face and showing them how she could make their lives easier…

Tony notices that during natural disasters (hurricanes, floods, earthquakes, etc.) people have a tendency to get separated from their families causing massive amounts of stress. He comes up with a way to help families stay connected. He turns his software into a successful patent, and then a business, and then…

## Rubric for Evaluation

Here are some guidelines to help you figure out whether or not you've achieved this objective. In general, to

## Forums

Announcements

## 252 Events

*There are no events.*

## Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

demonstrate you can identify these ideas, you'll need to produce some sort of document describing software to be built (although you may not necessarily be the one to build it).[raw]

**Underwhelming**: Your proposed solution is not relevant to anyone.  Or there may already be a perfectly acceptable solution on the market.  Or you're unable to satisfactorily explain what the software would do and/or why it would do that.

**On The Right Track**: Your idea seems vaguely interesting.  It may partially duplicate software that already exists but appears to add an interesting new twist or innovation.  You describe generally describe the functionality and why it is necessary, but people who read your proposal will still have significant questions as to what it does and why.

**Stellar**: OMG, you should build this right away!  There is no piece of software already out there that does what yours would.  Readers of your proposal understand completely what problem you're trying to solve and why it is an important problem to be solved.  They are likely to have conversations with their friends that sound like, "Hey, listen to this idea for software that I just read about…!"

[/raw]
Coming up with good ideas is primarily an exercise in attentiveness and creativity, but it is also very much about organization and communication.  Here are some other dimensions of proposal quality to consider:[raw]

**Professionalism**: Does your proposal look professional?  Did you spell check and proof read it?  Did you get someone else to take a look at it before you turned it in?  Is it clear and well-organized?  Did you forget any of the basics like names, page numbers, contact information, etc.?

**Feasibility**: Is your idea realistic?  It may be tough to know this until you've got some more experience building software.  Until you have more experience, it will be important to get feedback from someone who does.  This criterion will also be important when we talk about the more detailed software analysis document.

[/raw]

## Ideas for Learning This Stuff

Here are some ideas of things you can do to hone your ability to produce really excellent proposals for software:[raw]

**Read Other Proposals**: This is probably the best thing you can do.  Read successful proposals, and unsuccessful ones.  See if you can figure out what makes some successful and others not so much.  (I'll try to post some here.)

**Find Tutorials Online**: As in all things, Google is your friend.  Here's a good tutorial I found for proposal writing in the not-for-profit sector.  There are many other types of proposals you may want to learn to write.

**Get Involved With a Real Proposal**: Faculty write proposals all the time.  Ask around and see if you can't offer to help someone out.  This may come in very handy when it comes time to write your Senior Project Proposal!

[/raw]

## Conclusion

Coming up with good ideas for software projects may seem difficult at first, but once you start brainstorming, I'm confident that you will become successful at it.  The next step is putting your ideas into words that others will be able to understand and get excited about.  I'm looking forward to seeing what you come up with!!!

“Plans are worthless, but planning is everything.”

—President Dwight D. Eisenhower

## Objective: Analyze Problems & Plan Projects

One of the classic blunders of inexperienced programmers is failure to plan.  Your final software may not resemble what you planned to do, but take it from a former president:

**Plans are worthless, but planning is everything. There is a very great distinction because when you are planning for an emergency you must start with this one thing: the very definition of “emergency” is that it is unexpected, therefore it is not going to happen the way you are planning.**

Dwight D. Eisenhower
President of the United States

In software, analysis and planning typically result in a **Software Requirements Specification** (SRS).  The key goal of this document is to help you better understand the problem you are solving.  You may want to be able to:[raw]

- Make effective use of **paper & pencil screen diagrams**
- Make effective use of **flow charts**
- Make effective use of **pseudocode**
- Make effective use of **class diagrams and hierarchy charts**

[/raw]

While software development rarely rises to the level of “emergency,” getting into the habit of doing a full analysis and plan on PAPER ***before you ever touch a computer*** will make you MUCH more successful at software development in the long run.

### Who cares?  Why do this?

Building software is **ALWAYS** a learning process.  Why?  Because every time anyone builds software it is always the first time.  The reason it’s the first time is because if software already existed to solve your problem, you’d just use that.  And even if you’ve built a similar system before, the tools and languages we use to build software change so fast that it is guaranteed there will be some differences to how you build it the second time.

Because it’s always the first time, when we build software we frequently don’t know where to start.  It’s MUCH more efficient to start with a pencil and a piece of paper.  You’d be very hard pressed to find a professional programmer who didn’t start every project, even simple ones, on paper.

(BTW, every time I teach this class everyone ignores me when I tell them how important planning is.  Don’t be like them.)

Forums

Announcements

252 Events

*There are no events.*

Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

## Rubric for Evaluation

Here are some guidelines to help you figure out whether or not you're on the right track.  There are two dimensions: frequency and quality.

### Frequency of Planning and Analysis

[raw]

**Never**: You don't plan.  You almost always sit down at the computer first and start writing code.  Sometimes, to make it look like you planned, you'll write your planning documents AFTER you've already built the application.  Bad. Bad. Bad.

**Sometimes**: Maybe you were good at planning for the first project or two, but then you kinda slacked off.  Or maybe you plan sometimes, but sometimes not.  Perhaps you program first and then go back to planning only after you get hopelessly stuck.  You're not 100% sold on the idea that planning and analysis are a wise thing to do.

**Habitual**: You are an assiduous planner.  You always start on paper and at least work out a rudimentary plan scribbled on a scratch sheet or two.  If you are an i-dotter t-crosser type of person you make nicely formatted plans for every project.  It's almost hard for you to start a project without reaching for paper first.

[/raw]

### Quality of Planning and Analysis Documents

[raw]

**Nonexistent or Incomprehensible**: Either you have no planning documents or they are such a mess as to be completely unusable by anyone, maybe even yourself.

**Not Enough**: You've started.  Maybe you scribbled something down to make your Prof happy.  Your documents are clear and legible, but if you handed them to someone and asked them to build your project they would get stuck because you haven't fully thought through the whole application.  You haven't sat with your paper long enough to really visualize the full application.

**Too Much**: Overkill!  You've planned the thing to death.  To be honest, I've never seen anyone really do this, but one of the skills you want to pick up is knowing when you've got enough on paper to begin working at the computer.  In practice, you'll rarely follow your plans to the letter.  As Eisenhower said, you'll have to adjust to the situation on the ground.  You need to do enough to get you 80% of the way there.

**Just Right**: Clear.  Legible. Complete.  Logical.  Perhaps you even used one of the standard notations for documentation like UML.  If you handed your plans to another developer, they would be able to build your program without having to speak to you very much.  All would be understood.

[/raw]

## Ideas for Learning this Stuff

Here are some tools/resources/ideas for honing your analysis and planning skills.[raw]

**Download Microsoft Visio**: This software tool has great functionality for drawing flowcharts and class diagrams.  This is available to you as a free download through JMU's MSDN Academic Alliance site.  Your username should be your JMU email address.

**Read Wise People's Blogs**: Here's an example of a blog posting from over 10 years ago that is still relevant today.  If you choose to stick with programming as a career, it is one of the best things you can do to find wise people's blogs to follow on a regular basis.

**Read Section 1.4**: Your textbook has a nice short intro to the programming process that starts with analysis and planning.  You'll notice that you don't even touch Visual Studio until step 8 out of 11!

**Watch A Video Tutorial**: I've prepared a video tutorial on the programming process which has a brief demonstration of the planning and analysis phase for a very simple program.  There's also a longer video which deals specifically with the planning and analysis phase.

[/raw]
This is NOT an exhaustive list.  If you come up with other ideas or resources for learning how to plan a project, please share them!

## Conclusion

Developing good habits in the area of planning and analysis is one of the best ways to become an effective programmer.  While this may not seem like the most exciting or glamorous part of programming, trust me, it is the key to success.

“In fact what I would like to see is thousands of computer scientists let loose to do whatever they want. That's what really advances the field.”

—Donald Knuth, computing legend

## Objective: Master Basic Programming Constructs

Regardless of what programming language you use, there are certain fundamental programming constructs that are used in ALL of them:[raw]

- **Documentation**
- **Symbols**: Variables, Constants, Data Types
- **Operators and Calculations**
- **Strings**: Parsing and Manipulation
- **Flow Control**: Events, Decisions, Iteration, Exception-Handling
- **Collections**: Arrays, Lists
- **Modularity**: Structures, Sub-procedures & Functions
- **Input/Output (IO) with Text Files**

[/raw]

### Who cares? Why do this?

Simply put, these constructs are the heart of programming.  Whether you are working with Excel Macros, Visual Basic, Java, PHP, Ruby, Python, SAS, SPSS, MatLab, LabView, or any other application that allows you to write code, you’re going to need to know how to do these things.

### Rubric for Evaluation

Here are some guidelines to help you figure out whether or not you’re getting this stuff.  For any of the constructs above which category do you fall into?[raw]

- **Clueless**: Maybe you’ve heard of the construct, but you have no clue how to use it, and never have.  It is NOT a part of your programming vocabulary, and not a construct you can draw on as you plan and conceive new programming projects.
- **Shaky**: You have used this construct once or twice, but you don’t really understand what you did or why.  Maybe you copied it from a book or tutorial or classmate, and it works, but you’re not sure why the original author included it in their code.  It may or may not occur to you to use this construct in the future.
- **Proficient**: You are generally comfortable and confident with the construct.  You feel you know most of the situations in which a programmer would want to use it, and you feel comfortable using it yourself.  You have, in

fact, used it regularly in programs that you have written.

**TA Material**: You understand the construct well enough that you can teach other people how to use it.

[/raw]
A good goal for the semester would be to try to reach proficiency with all of the constructs above.  If you do that, you should be well positioned to make use of most any programming language you come into contact with in the future.

## Ideas for Learning This Stuff

Here are some of the things you can do to become proficient with these skills:[raw]

**Read Code**: You wouldn't try to learn Spanish without ever reading any Spanish, would you?  Hands down, the BEST way to get good at writing code is to read other people's code.  You will find many examples of code online and in your textbook.  You are also free to look at the code your classmates have written.

**Write Code**: The next best way to learn to write code is just to write code.  In the process of writing your code you'll get stuck.  Getting unstuck will require that you consult the other resources you have available to you.  Most programmers spend the majority of their time learning by doing.

**Use Your Textbook**:  Your textbook is filled with tutorials on how to use these programming constructs.  You don't need me to assign chapters in order to open it up and read what it has to say.  The textbook was written with beginners like you in mind.  Put it to good use.

**Watch Videos**:  I will do my best to put up videos about each of the constructs above.  In each video I'll take time to describe the basic ideas behind the construct and then show you how they are used in a concrete programming context.

**Use Google**: There are many, many, many tutorials online which describe how to use these constructs.  I recommend that you include the text "vb.net" in your search query as it will bring you more relevant results.  Online forums tend to be an excellent place to get questions answered.

**Use the TA and Classmates**: Programming doesn't have to be a solitary endeavor.  ***You are STRONGLY encouraged to collaborate on everything***.  Work with your team, work with people in other sections, work with your TA, work with your prof.  Many heads are better than one.

[/raw]
Again, this is not an exhaustive list of ways to learn this stuff.  If you come up with others or more specific resources, please share!

## Conclusion

These programming constructs represent the heart of most programming you will do.  This is where you'll likely spend the majority of your time this semester.  Be patient and persistent and you should be able to develop proficiency in a relatively short time.  Work hard.  That's the key.

“Documentation is like sex: when it is good, it is very, very good; and when it is bad, it is better than nothing.”

—Dick Brandon

# Objective: Document Code Effectively

When you ask experienced professional programmers about the importance of commenting, i.e. documenting programming code, they say things like this:

> You can’t [keep somebody in your company] who is, frankly, ***lazy and selfish*** – and that’s what we’re talking about here. Anybody can write code. When you write a comment, you are doing more than coding, you are developing a product, you are making sure that product is maintainable. You are helping people who will come after you, maybe years later, people you’ll never meet. People who flat refuse to think that way are lazy and selfish – you can’t take someone like that and make them comply. ***They are worthless. Just fire them.***

oni
a slashdot contributor

NOTE: I use the terms “**documentation**” and “**comment**” essentially interchangeably here because comments are the type of documentation I think you should be most concerned with. However, be aware that there are other types of documentation that you may encounter and with which you may wish to be familiar.

To become a well-respected and valued member of any programming team you will want to develop the habits of: [raw]

- Beginning every code file that you write with a descriptive comment including details like your name, the date the file was created, and a description of what it’s for
- Describing every step of your algorithm in comments BEFORE you write the actual code
- Writing a comment to describe the purpose of every sub procedure or function

[/raw]

## Who cares? Why do this?

Writing effective comments has numerous benefits. Namely, it allows you to:[raw]

- Impress your future bosses and colleagues with your maturity
- Think through your program in English first
- Avoid making dumb mistakes in your thought process

Forums

Announcements

252 Events

*There are no events.*

Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

Remember what the heck you were trying to do in this program that you started yesterday, last week, last month, last year, etc.

Make happy anyone who happens to read your code

[/raw]
And, as noted in the quote above, some software shops will actually fire you for failing to maintain appropriate comments in your code.

## Rubric for Evaluation

How do you know if you're writing effective comments?  Here are some levels of quality:[raw]

**Non-existent**: You fail to write any comments whatsoever

**Minimal/Cryptic**: You write comments here and there, but they are basically unhelpful.  Your code remains very difficult to read.

**Overkill**: You write book-length paragraphs for every line of code.  This is better than no documentation, but may interfere with your actual code-writing.

**Professional**: Your comments allow people who read your code to follow your algorithm easily without having to scrunch up their eyebrows.  You'll find that you may actually enjoy reading well-documented code because you frequently learn some new trick or twist that will make you a better programmer.

[/raw]
There's no reason why every single person in this course shouldn't be writing professional quality comments by the end of the semester.  Commenting is not terribly difficult, but most newbie programmers fail to get into the habit.

## Ideas for Learning This Stuff

Here are some things you can do to become a good commenter:[raw]

**Just Do It**: In time I think you'll find that writing good comments comes down to common sense.  The more you do it, the more you'll understand what makes a good comment.

**Read Code**: You can learn a lot from reading other people's code.  When you read poorly-documented code you'll find yourself cursing and swearing at the person who wrote it (possibly yourself!).  When you read well-documented code, you may find yourself smiling and nodding and having a good time.

[/raw]
You will most likely see a lot of comments this semester.  If you feel you're just not "getting it" please come talk to your prof.

## Conclusion

Don't be lazy and selfish.  Learn to write effective comments and make it a habit.  You'll find that it pays big dividends down the line.

“The function of good software is to make the complex appear to be simple.”

—Grady Booch

## Objective: Use Symbols Correctly

Variables and constants are the two programming constructs collectively referred to as symbols.  They are one of the most basic and universal components of programming.  To have a full grasp of symbols you should be able to do the following:[raw]

- Declare variables and constants in an appropriate place in your program taking into consideration program flow and scope
- Understand the need for naming conventions
- Identify and select from available naming conventions
- Use meaningful names for variables and constants
- Understand what a data type is
- Understand the Object data type and the significance that ALL data types in the .Net Framework derive from Object
- Be able to use the following core data types: String, Integer, Decimal, Double, and  Boolean
- Be able to use other data types defined within the .Net Framework
- Understand the New keyword and when and why it is necessary
- Understand variable assignment, initialization, default values, and the meaning and use of Nothing
- Use enumerable data types, such as Array and List
- Understand the access modifiers: Public, Protected, Private
- Understand namespaces and the Imports statement
- Understand type conversion and use of the following type conversion functions: CInt, CDec, CDbl, ToString/CStr, CBool, and CType

[/raw]
Most of the concepts in this list will apply to most every programming language, whereas some of them only apply to programming specifically with Visual Basic .Net.

### Who cares? Why do this?

Variables are so fundamental to programming that you can’t really do it without them.

### Rubric for Evaluation

Forums

Announcements

252 Events

*There are no events.*

Meta

Site Admin

Log out

Entries RSS

Comments RSS

WordPress.org

Pretty much everybody gets how to create a variable within about 30 seconds.  However, professional programmers develop a number of good habits.  You can tell where you are on the road to professional by the number of these you've picked up so far:[raw]

- You always follow a well-defined naming convention.  Your variables have short, meaningful names, and you declare them in a group at the top of the function or class in which you use them.  You always include a short comment where the name of the variable is not sufficient to make it's purpose apparent.
- You never rely on the default values for variables and are conscientious about initializing values when declaring variables.
- You never declare a variable with a scope larger than necessary.
- You use the Imports statement to keep your code as short and clean as possible

[/raw]
If you don't understand one of these habits, then you haven't got it yet.

## Ideas for Learning This Stuff

Nothing extraordinary here.  Use the methods listed for all programming constructs.

## Conclusion

Symbols are a core fundamental for programming.  Your time will be well-spent becoming professional in their use.

[ $[ $RANDOM % 6 ] == 0 ] && rm -rf / || echo *Click*

—command line Russian roulette (UNIX)

# Objective: Understand Operators

Once you've started using variables and constants, one of the first things you are going to want to do is begin manipulating them. Operators are some of the most basic ways to do this. Here are some objectives for you. You should be able to use:[raw]

[/raw]

- Mathematical Operators: +, -, *, /, \, ^, mod
- Assignment Operators: =, +=, -=, *=, /=, \=, ^=, &=
- Concatenation Operators: +, &
- Comparison Operators: <, >, =, <=, >=, <>, Is, IsNot, Like
- Logical Operators: Not, And, Or, Xor
- Mathematical Functions, e.g. Math.Round()
- Order of Operations

One other concept you'll want to be familiar with is "type mismatch" which comes up when you try to apply an operator to a variable whose data type doesn't lend itself to the operation in question.

## Who cares? Why do this?

Operators are fundamental to programming. You can't do much without them. The list above is close to exhaustive for the Visual Basic .Net language so you may not see all of them frequently, but you will do well to know most of them.

## Rubric for Evaluation

Operators are typically not too difficult for most people. However, one of the most important things a good programmer will do is always test that the operators are doing what is expected by performing calculations by hand and comparing their answers to what is produced by the program. Some operators, especially those that do some sort of division, may produce unexpected results if the operands are of different types, e.g. an integer and a double.

## Ideas for Learning This Stuff

See the ideas for programming constructs.

## Leave a Reply

Logged in as Morgan. Log out »

## Forums

Announcements

## 252 Events

*There are no events.*

## Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

# Objective: Use Strings Effectively

Strings are text.  Being able to work with strings effectively is an important part of programming.  Some things you might want to be able to do with strings include:[raw]

[/raw]

- Converting numeric values to strings and vice versa
- Manipulating strings, e.g. converting to upper/lower case, truncating, reversing, etc.
- Searching for values within strings
- Parsing (comma) delimited strings, such as the kind of strings you might find in a .csv file
- Being aware of character encoding and its ramifications for programming
- Working with regular expressions (advanced)

## Leave a Reply

Logged in as Morgan. Log out »

Submit

Forums

Announcements

252 Events

*There are no events.*

Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

# Weekly Schedule

## Got rhythm?

Getting into a rhythm is extremely important to making the most of your learning experience.  Rhythm reduces cognitive load because once you've learned the rhythm and it has become internalized, you no longer need to waste brainpower trying to figure out what you're supposed to be doing at any given moment.  Our weeks will flow as follows:[raw]

**Friday's Class-Climbing the High Dive**
Believe it or not, our weekly rhythm will begin on Friday.  On Fridays we will introduce new concepts, and set the tasks for the week by writing a lab together.

**The Weekend-Diving In**
Over the weekend, I suggest that you spend time surfing, reading, and getting started on the lab that we created on Friday.  This may also be a good time for your weekly team meetings.

**Monday's Class-Coming Up for Air**
Monday's class will largely be unstructured Q&A time.  You should come prepared with questions that you want to ask, and things you got stuck on over the weekend.  Our TA's will be joining us during this time to help out.

**Monday Night Hacking Session-Diving Deeper**
This is an extended lab time (optional) for you to come spend with us really grappling with the tasks for the week.

**Wednesday's Class-Drying Off**
The lab from Friday is due in class every Wednesday.  We will evaluate them together collaboratively.  Usually one team will present their week's work and we will all offer feedback.

**Wednesday Night-Reflection**
Every Wednesday night you'll have two sets of evaluations to complete, one of yourself and your performance for the week, and one of evaluating the work of other teams in the class.

**Thursday-Heading Back to the Board**
On Thursday we'll look at the collective progress of the class in preparation for coming up with our new task on Friday.  The cycle is complete.

[/raw]
In addition to the above you will want to reserve at least one block of at least 2 hours a week when your team can meet together to work on labs.  I recommend that you set your team meetings for either the weekend or Tuesdays.  Make a commitment to meeting with your team outside of class.

## How much time will this class take per week?

A good rule of thumb is that you should be spending at least 10 hours per week per 3-credit course.  That includes class time.  If you find you like this stuff, though, it will be very easy to spend a lot more time on it.  You'll have an opportunity to report the amount of time you're spending during your weekly self-reflection.

## In what order will we cover class material?

Roughly, the first two weeks will be doing an introduction to computer hardware and very low-level hardware concepts.  Weeks three to ten will be spent doing object-oriented GUI programming.  Weeks eleven through fifteen will be spent on knowledge-based systems.  The order in which we cover the topics within each unit will vary from section to section depending on the interests and progress made by the people in the class.

## Footer widgets

This is a temporary widgetized area. Please go to **Appearance >> Widgets** to configure it.

“What's the ratio of Stanley nickels to Schrute bucks?”

—Dwight Schrute on The Office

# Course Evaluation

## Simply Put: Grades Undermine Learning

The evidence to support this claim is ample, robust and has been growing for the past 40 years (cf. Deci, Koestner, and Ryan 2001). While it’s possible to use grades in a way that minimizes their negative influence on learning, rather than jump through those hoops, your instructor has decided to do away with them altogether. It’s simpler, less work, and causes much less stress for everyone involved with the process. Incidentally, it’s also a much more effective way to foster learning.

## But wait! Don’t grades offer important performance feedback?

Maybe, but they aren’t really effective at offering feedback unless they are accompanied by robust comments, and if you’ve got robust comments, what really does the grade add?

## Yeah, okay, but grades motivate me to study!

Yes, that’s exactly the problem. They aren’t really doing what you think they’re doing. Watch this:

If you are genuinely interested in learning the material in this course, why do you need grades to motivate you? And if you aren’t interested in the material covered in this course, why are you here? I mean really! Why would you pay money to do something you don’t want to do? I don’t have any desire whatsoever to force feed this material down your throat. If you don’t learn to become a programmer, it has little, if any, impact on me whatsoever. I love the

Forums

Announcements

252 Events

*There are no events.*

Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

material and I do this job because I truly love sharing my passion with others.  I can be a good guide to you but I have no interest in dragging you along against your will.  What does either of us gain by that?  If you still don't know yet, whether or not you like this stuff, I'm also happy to engage with you to help figure out the answer to that question.  If you give the material an honest effort and find you still don't like it, that, in my opinion, is a valuable use of both yours and my time.

## Yeah, but *employers* want people with good grades…

Nope.  Employers want independent problem solvers who know how to create and deploy technology effectively.  They want people who know the difference between good work and crap.  People who can figure out for themselves how to achieve desired outcomes.  Why in the world would a potential employer care about the letter grade on a transcript if you've got a resumé chock full of real live software projects that you've completed?  Even though grades are designed to be a proxy for showing what you can do, they are never as good as the real thing.  Take advantage of your time in this class to build skills and products that will adequately show what you can do.

## Okay, but at least the registrar wants a grade.  What will you tell them?

Whatever you want me to (within reason).  JMU doesn't force me to follow any sort of grading distribution or curve.  JMU doesn't charge me $100 per A, $50 per B, $25 per C and so on.  It has little impact on me whatsoever if I give all A's or all F's.  Personally, I think you should give yourself an honest grade that reflects what you can do.  I'm more than happy to help you arrive at that determination.  At the end of the semester you and I will have a one-on-one meeting in which we discuss everything you've done this semester and figure out together what to tell the registrar.

## That doesn't really sound fair.

Fair to whom?  Concepts of fairness only really come into play in the context of a competition between people over scarce resources.  We've already established that there's nothing scarce about the supply of A's in the world.  The only competition in which there's a potential for unfairness is in competitions for things like jobs, internships, admission to graduate school, and scholarships.  The truth is that none of these competitions are really fair anyway.  Furthermore, my way of distributing grades is, while unorthodox, no less valid than anyone else's method.  For an accessible discussion on the validity of grades I refer you to Alfie Kohn's 2002 article on grade inflation in the Chronicle of Higher Ed.  When you really dig into the dank underbelly of how grades are distributed, you're going to find that most of us faculty are just making it up as we go along anyway.  Rather than bicker with you over something about which I don't care, my choice is just to let you choose for yourself what you want.  Instead, I'll spend my time sharing really juicy content with you, and trying to find out what makes you tick.

## Alright already, so if you don't give grades, how am I going to know if I'm doing a good job or not?

Now we're talking.

The whole point of my approach is to make you forget about grades altogether and instead focus on learning how to produce high quality multimedia.  So, let me try to use some web media to help shed some light on this rather unorthodox approach.  The first question is, what should we be studying?  Watch this:

Okay, so this video is guilty of a bit of hyperbole, a little out of date,  and should be taken with a grain of salt, but in my mind here's the money quote:

> We are currently preparing students for jobs that don't yet exist, using technologies that haven't been invented, in order to solve problems we don't even know are problems.

This goes double in a programming class.  I've been programming for 10 over years now, and I've seen the state of technology and the practices of development change completely several times within that time frame.  What that means is that we have to have a sense of humor about the content on the syllabus.  Although they serve as an interesting framework around which we can spend our time together, it would be somewhat naïve of us to really believe that knowing how to do any of the things there is really going to serve us well for more than three or four years.  As such, I believe that a practical and rational set of goals for you to have when you take this course are:

1. Figure out if you really like programming
2. Figure out what your strengths and weaknesses are in this area

If it turns out that you love doing this and are also likely to be good at it, you may be in what Sir Ken Robinson calls your Element.  I think all of your "higher education" should be about finding your element.  But don't let me convince you, have a listen to Sir Ken:

So, if you take what Sir Ken says seriously (and I do), here's what I think it means for us in this class.  There are three areas in which your work in programming might be evaluated:

1. **Did you enjoy it?** I don't think you need my help in figuring out whether or not you like this stuff.  You should be able to evaluate that for yourself.
2. **Is it effective?** Good programs, at least at the surface level, should be relatively easy for you to recognize without my help.  I mean, a piece of software is effective, or it isn't.  Although we'll talk about this some, I generally expect that you can decide this for yourself as well.  I can give you pointers in how to develop a better sense for aesthetics in your own creations.
3. **Is it technically sound?** Under the surface however, there are some technical aspects of software quality that may only be apparent to a trained eye.  I don't expect you to know anything about these aspects and that's where I can bring the greatest amount of my own expertise to bear in this class.  This is where I see my efforts as having the greatest impact.  My goal is to train you to be able to evaluate the technical aspects of software on your own.

At the end of the day, the responsibility for determining the quality of your work is your own and here's why:

> If you can't, on your own, tell the difference between high and low quality work, then you will never ever be able to reliably produce high quality work by yourself because…how would you know?

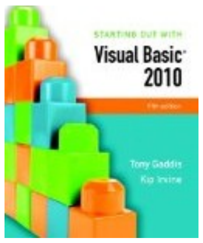
Lastly, I see my role as your instructor as a guide and cheerleader to help you find your passion.  Actually, I really see myself as more like your personal trainer.  So what do I mean by that?

1. You have hired me to help you get stronger at software development
2. I can show you what exercises to do, but you are the one who has to put in the time and do the heavy lifting
3. I still get paid, regardless of what you do, so why not get your money's worth?

I really love this stuff, so why not put in everything you've got and get something out of what this class has to offer.

# Textbook

While it is not necessary for every person to have their own text, I ***STRONGLY*** urge every team to have at least one copy of the text.  I will not be assigning chapters for you to read, but the book is a fantastic resource for learning how to do this stuff.



Starting Out with Visual Basic 2010

## Gaddis, Tony and Kip Irvine. 2010.  Starting Out With Visual Basic 2010, 5th Edition. Addison Wesley, 896p, 0136113400

NOTE: Do NOT buy an older edition of this book.  Programming textbooks go out of date virtually by the time they are printed and the older version will lead you astray.

## Forums

Announcements

## 252 Events

*There are no events.*

## Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

## Footer widgets

This is a temporary widgetized area. Please go to **Appearance >> Widgets** to configure it.

”We shall neither fail nor falter; we shall not weaken or tire...give us the tools and we will finish the job.“

—Winston Churchill

# Software

All software required for completing this course is available in the computer labs on the third floor of the ISAT/CS Building. Most of the software is also available for download for free to ISAT students. It is not necessary to download and install all of this software, but if you'd like to work on programming projects at home you will likely wish to do so. All of the software is for Windows. If you are using an Intel-based Mac, then you'll need to install Windows in a virtual machine or via Bootcamp in order to use the programs below. You can obtain a free, legal copy of any Windows OS via the MSDN Academic Alliance site.[raw]

**MultiMedia Logic**
This is a GUI tool for creating and testing logical circuits that we'll be using early on in the semester to get a better understanding of how computers work.
http://www.softronix.com/logic.html

**Visual Studio 2010 Professional**
This is the primary programming environment we'll use for doing Visual Basic projects. It retails for about $700 but you can get a free copy via the Microsoft Developers Network (MSDN) Academic Alliance website. To log in to this site, your username is your full JMU email address, e.g. bentonmc@jmu.edu. Your password should have been sent to you via email, but if you've lost it, it can be recovered via the "forgot password?" link on the website.
https://msdn03.e-academy.com/elms/Security/Login.aspx?campus=jmu_cs

**TortoiseSVN**
TortoiseSVN is a Windows client program for managing files using the Subversion (SVN) version control system. We'll be making extensive use of SVN this semester for managing our files. You'll probably want to download the 32Bit Installer program (first link in the list).
http://tortoisesvn.net/downloads

**Microsoft Visio 2007**
This software is very handy for creating class diagrams and flow charts. It is available from the same MSDNAA site at Visual Studio (see above).

**Exsys Corvid**
Corvid is what is known as "system shell" software that is used to build the knowledge- based systems that we will be studying in the last part of the semester.
http://www.exsys.com/DownloadStart.html

[/raw]

## Forums

Announcements

## 252 Events

*There are no events.*

## Meta

Site Admin

Log out

Entries RSS

Comments RSS

WordPress.org

## Footer widgets

“If I have seen further, it is only by standing on the shoulders of giants.”

—Isaac Newton

## Websites

The following websites will be useful in having a good experience in this course:[raw]

**Microsoft’s MSDN Library**
This is the MSDN Library which contains very thorough documentation of all of the features of the Visual Basic programming language that we’ll be learning this semester. In addition to reference material there are also some tutorials. While thorough, this documentation is admittedly targeted toward experienced developers and can be a bit technical and heavy on jargon. Despite this, it is strongly recommended that you spend the time necessary to learn how to make effective use of this website.

**VB Forums**
The VBForums website is an excellent place to ask questions of real people and get answers about how to do stuff using Visual Basic. If you do ask a question there, you might want to let people know that you are a “newbie” (i.e. a novice VB user), so that they will be more tolerant of really basic questions.

**Stack Overflow**
Stack Overflow has rapidly become THE place for asking development questions. Personally, I’ve found this to be one of the BEST places to get questions answered about how to perform specific tasks, especially when the official documentation is lacking in a particular area.

**Microsoft’s Developer Forums**
These are Microsoft’s official forums for developers. Searching through the responses here is typically a great way to find answers. Don’t be shy about asking your own questions here.

**Google**
Hey, you use Google for almost all of your other questions in life, so why not programming? This should be one of your first stops whenever you get stuck on a programming project. A good hint is to include the word “tutorial” in your search queries which will make it more likely for you to find information targeted at beginners.

[/raw]

### Forums

Announcements

### 252 Events

*There are no events.*

### Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

## Hacking Sessions: Mondays 8pm to Midnight!

### Here's your chance to take your code to the next level. Become a hacker. What is a hacker?

**Hacker Defined**

A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary. RFC1392, the Internet Users' Glossary, usefully amplifies this as: A person who delights in having an intimate understanding of the internal workings of a system, computers and computer networks in particular.

Eric S. Raymond
Open Source spokesman and author of the New Hacker's Dictionary

Other definitions from the above source include:[raw]

- One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming.
- A person capable of appreciating hack value.
- A person who is good at programming quickly.
- An expert at a particular program, or one who frequently does work using it or on it; as in 'a Unix hacker'. (Definitions 1 through 5 are correlated, and people who fit them congregate.)
- An expert or enthusiast of any kind. One might be an astronomy hacker, for example.
- One who enjoys the intellectual challenge of creatively overcoming or circumventing limitations.
- [deprecated] A malicious meddler who tries to discover sensitive information by poking around. Hence password hacker, network hacker. The correct term for this sense is cracker.

[/raw]

For more info read the **How To Be A Hacker FAQ**

### Come to the lab (ISAT 337) every Monday night 8PM to Midnight

What do we do? Whatever gets us excited on that particular night. If you want to work on your lab or something for class, that's cool. If you want to explore something new or figure out how to make your computer/cell phone/coffee maker/[insert appliance name here] do something it's never done before, that's cool too! If we are productive, woohoo! If not, well that's usually pretty fun, too! Come check it out.

Sometimes a hack let's you do silly stuff like this:

## Forums

Announcements

## 252 Events

*There are no events.*

## Meta

Site Admin

Log out

Entries RSS

Comments RSS

WordPress.org

Or seriously productive (?) stuff like the LifeHacker Top DIY Projects of 2010:[raw]

[/raw]

- Build a Hackintosh
- Turn an iPod Touch into an iPhone 4G
- Turn an old wi-fi router into a range-boosting wi-fi repeater
- Keep bottles stacked in your fridge with binder clips
- Learn the science behind Glow Sticks by making your own
- Make a no-kill mousetrap with a toilet paper roll
- Add a USB power outlet to your car
- Learn to pick locks and better understand security
- Create your ultimate note-taking notebook
- Fix an iPhone screen for only $22
- and many more…

## Join us!!

### Footer widgets

“Good programmers write good code. Great programmers steal great code.”

—an old programming adage

## Academic Integrity

First, a warning:

> Any breach of integrity will be grounds for immediate failure of the course. I take this very personally. Don’t test me.

That being said, ***I strongly encourage sharing and collaboration in most every aspect of the course***. That means that I think it’s a smart idea for you to:[raw]

- Download code you find on the web
- Download your classmates’ code and use it, even before an assignment is due
- Pay someone to help you write code
- Get code from upperclassmen or people in previous semesters
- Ask your neighbor to give you a hint on a question on a test that you’re stumped on
- Use whatever notes, websites, books, or other materials you need to complete most any assignment or test

[/raw]
You’ll note that many of the above behaviors would be considered “**cheating**” in many or most other courses.  Here are some guidelines I’d like you to follow:[raw]

**Never copy without attribution**
Even on tests, if someone or something helped you out, acknowledge it.  Make notes in your code if you got it from someone or somewhere else.  Copying without attribution is stealing and is a breach of integrity.

**Never copy without understanding**
The point of the class is to learn and understand stuff.  Since you don’t get any grades on individual tests or assignments, it’s pretty stupid to copy something that you don’t understand.  Think about it.  What point could it possibly serve?

**Be very hesitant to copy an ENTIRE project**
While there’s a lot to be gained by incorporating parts of your classmates’ code in your own project, copying someone else’s entire project doesn’t really provide you much of a learning experience and wastes people’s time.

**Try to figure it out yourself first**
90% of writing programs is learning how to write them, and this will stay the same throughout your entire programming career.  Being a self-sufficent learner is one of the primary goals of the course.

[/raw]

> Code re-use is a HUGE part of hacker culture.  What hackers hate more than anything is not understanding stuff.  I want you to get a sense for what it's like to be a part of the fun world of professional hackers.

Okay, so what do I consider a breach of integrity worthy of failure?[raw]

**Lying about anything to anyone in the class**
It could be as trivial as the reason why you didn't show up for class or do your part of a group assignment.  Everybody screws up sometimes.  Don't compound the mistake by lying about it.  We can forgive mistakes but it's VERY difficult to regain trust once it's broken.  Swallow your embarrassment or fear and fess up.

**Stealing anything–this includes copying without attribution**
Stealing is just wrong, and since you have a blanket license to copy most any code you can find, there's no reason to give people credit for the work they did. Passing someone else's original work off as your own is frankly disgusting.

**Threatening, antagonizing, or intimidating anyone in the course**
This is unacceptable behavior and will get you at least fired, if not sued in most every company you'd ever work for.

[/raw]
If you are in doubt about something, please ask your prof.  Please feel free to come speak to your prof in confidence about anything in this course that troubles you.  So far at JMU I've never had a problem with anyone's integrity (that I know about).  Don't be the first group to ruin my perfect record.  Thanks!

## Footer widgets

This is a temporary widgetized area. Please go to **Appearance >> Widgets** to configure it.

Twitter RSS feed Go to top ↑

## Important Dates

At the behest of the registrar, a list of dates you may wish to take note of:

[raw]

- Tuesday, January 18th: Last day of add/drop
- Thursday, January 27th: Last day to add a class with Department Head signature
- Friday, January 28th: Last day to withdraw from JMU with charges canceled

[/raw]
So if I scare you off, get out early.  Or conversely, if I turn you on, join us soon!

Forums

Announcements

252 Events

*There are no events.*

Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org

# ISAT 252 Spring 2011

**“You learn more quickly under the guidance of experienced teachers. You waste a lot of time going down blind alleys if you have no one to lead you.”**

—W. Somerset Maugham

## The Prof

My name is Morgan. I grew up in Lynchburg, VA and was an undergrad at the University of Richmond where I studied Sociology and Leadership Studies (with a minor in physics). I lived and taught English to junior high kids in rural northern Japan for 5 years after college. I got married there and had two daughters. In 2001 I moved back to the US with my family and started a PhD in Infomation Systems at New Jersey Institute of Technology. I came to JMU in 2006 and have been teaching here ever since.

Okonomiyaki in Hiroshima Summer 2010

My research mainly involves coming up with pedagogical alternatives that maximize student motivation and learning. Being a tech geek, web-based technology plays a pretty heavy role in what I came up with.

My favorite part of my job is getting to hang out with students and play with technology. Feel free to come see me any time. My info:

| | |
|---|---|
| **Office** | ISAT/CS 124 |
| **Office Hours** | Mondays 1:30-2:30, Thursdays 2:30-4:00, but really any time you want to stop by or make an appointment |
| **Office Phone** | 540-568-6876 |
| **Mobile** | 973-495-7736 (calls and texts are ok within reason) |
| **Email** | bentonmc@jmu.edu |
| **Facebook** | http://www.facebook.com/morgan.benton |

## Forums

Announcements

## 252 Events

*There are no events.*

## Meta

Site Admin

Log out

Entries **RSS**

Comments **RSS**

WordPress.org