**CSCI 1470.3 Classroom Notes and Demonstrations**

8/27/2025

**The Computational Paradigm**

by K. Yue

**1. Paradigm**

* Every discipline has a paradigm: targets, objectives, considerations, bodies of knowledge, methodologies, standards, etc.

**Example #1**: In pure mathematics (as opposed to applied mathematics)

* An important objective to prove theorems which are true under a set of axioms (statements assumed to be true without proof).
* Methodologies: theorem proving, ...
* Quality considerations:
  + primary: truth or correctness.
  + simplicity and beauty.
* ...

E.g., the Pythagorean theorem: the sum of the squares of the two shorter sides (legs) of a right-angled triangle is equal to the square of the longest side (hypotenuse)

* Axioms: Five basic axioms of Euclidean geometry.

A triangle with a mathematical equation

AI-generated content may be incorrect.

* A mathematical conjecture is a statement that is believed to be true based on observations and patterns but has not yet been proven or disproven.

E.g., Goldbach's Conjecture (stated in 1742): Every *even* integer *greater than 2* can be expressed as the sum of two prime numbers. E.g.,

* 16 = 5 + 11
* 100 = 3 + 97 = 11 +89 = 17 + 83 = 29 + 71 ...

The Goldbach's conjecture is very easy to state but extremely hard to prove (one way or the other).

**Example #2**: History

* An important objective: understand the past to better comprehend the present and potentially inform the future.
* Methodology: a range of techniques and procedures for gathering, analyzing, and interpreting evidence from primary and secondary sources to construct a narrative and understanding of historical events.
* Some quality considerations:
  + Source selection, evaluation and preparation
  + Objectivity and bias mitigation
  + Methodology rigor
  + Narrative and communication

**2. Computing and software development**

* What is computer science?
* An example: Computer science (CS) is the study of computers and algorithmic processes, including their principles, their hardware and software designs, *their applications*, and their impact on society. ([Tucker et al., 2003](https://ftp.unpad.ac.id/orari/library/library-ref-eng/ref-eng-3/application/education/curriculum/k12final1022.pdf).)
* It is difficult to find another discipline more applicative than computer science.
* Computer professionals should be generalists.

**Software Development**

* primary objective: problem solving using software.
* methodologies: computer and software development techniques (that is what your computing degree means).
* One clear quality consideration: *correctness*, or does the program provide the right answer (expected output)?
  + Software testing is about increasing confidence in correctness.
  + A test case specifies:
    - input
    - expected output
  + A test case is passed when the expected output is equal to the actual output.

E.g., a Python to output the square of an input integer.

Test cases:

|  |  |  |
| --- | --- | --- |
| **Case #** | **Input** | **Expect Output** |
| 1 | 3 | 9 |
| 2 | 65 | 4425 |
| 3 | 0 | 0 |
| 4 | -65 | 4425 |
| 5 | 123456789 | 15241578750190521 |
| 6 | 'hello' | Error: input is not an integer. |

* Some important software development quality considerations:
  1. Correctness:
     1. Software verification:
        1. Software built according to the software specification. Is the software built right?
        2. Internal correctness
        3. e.g. does the program output the square of the input number right?
     2. Software validation:
        1. Software met the needs of the problem. Is it the right software?
        2. Constructing the right software specification: requirement analysis.
        3. External correctness
        4. E.g., do the users want the square of the input number?
  2. Performance
  3. Cost
  4. Meeting timeline
  5. Effectiveness
  6. Reliability
  7. Maintenance: software has a life cycle
  8. Security and privacy
* Do not just focus on internal correctness or software verification.

**3. Skills/Traits/Quality/Characteristics of software developers**

* People of course have different opinions.
* However, there are many commonalities.

**3.1 A simple non-scientific experiment**

Conducted a simple Google search for good software developer quality.

Four sources:

1. Coding Temple, 6 Key Qualities of Successful Software Engineers, <https://www.codingtemple.com/blog/6-key-qualities-of-successful-software-engineers/>.
2. MatchTech, 7 qualities exceptional developers possess, <https://www.matchtech.com/resources/7-qualities-exceptional-developers-possess>.
3. UMBC, How To Know If Software Development Is The Right Job For You, 8 Hard To Learn Software Developer Skills You Need, <https://www.umbctraining.com/software-developer-skills/>.
4. AlltheCode blog, Software Engineer Personality: 5 Traits For A Great Career, <https://allthecode.co/blog/post/software-engineer-personality>.

Results: quality/trait/characteristics of good software developers.

|  |  |  |  |
| --- | --- | --- | --- |
| **Coding Temple** | **MatchTech** | **UMBC** | **AlltheCode** |
| Technical Expertise | Solid technical knowledge | Curiosity | Curiosity |
| Problem-Solving Skills | Maintains an end-user focus | Creativity | Persistence |
| Attention to details | Fast self learner | Empathy | Ability to learn |
| Strong communications and collaboration skills | Strong communication skills | Patience, Perseverance and Problem-Solving | Focus |
| Continuous learning and adaptability | Dependent time and task management | Analytical | Pragmatism |
| Passion for the craft | A good team player | Detail-oriented and organized |  |
|  | 'Never say die' attitude | Adaptability |  |
|  |  | Communication |  |

* Although you can see their differences, some focus more on skills and qualities. Others focus more on traits.
* However, there are some common themes:
  1. Strong technical knowledge and expertise
  2. Strong problem-solving skills: analytical
  3. Good problem understanding: empathy, focus on end users, ...
  4. Strong communication skills
  5. Strong collaboration skills: good team play, dependable
  6. Continuous and fast self-learning
  7. Curiosity: passion for the craft
  8. Persistence: never day die, passion for the craft
  9. Organization and management skills
  10. Detail-oriented
  11. Adaptability

**Some Lessons**

1. We tend to focus on technical matters in our computing degrees.
2. Success in the computing profession requires much more than technical expertise. It will be increasing more so.
3. Personal traits, characteristics, habits, etc. take a lot of effort and time to develop.
4. Consider spending effort and time to cultivate them early and persistently.
5. Your university degree may give you an entrance ticket to the job market but it is not sufficient by itself.

Function:

E.g. f(x,y) = x + 2 \* y;

f(2,3) -> 8

Python code can be put together into modules

**Introduction to Turtle Graphics in Python**

by K. Yue

**1. Turtle Graphics**

* The Python turtle *module* is a *built-in* *library* (no need to download from an external source) that provides a way to create *graphics* and drawings.
* In turtle graphics, there are two major concepts:
  1. A window/screen/digit canvas for the turtle to move and draw.

A screen shot of a computer

AI-generated content may be incorrect.

* 1. A turtle with a pen that can move and draw (if the pen is down).

Turtle:

1. The turtle is:
   1. in a co-ordinate location (x,y), initially (0,0)
   2. facing a direction (initially right)
   3. holding a pen which can be up and down (initially down). If the pen is down, the turtle draws while moving.
2. There are functions to control the turtle. Two popular ones are:
   1. forward(distance): move the turtle forward by a distance. If the pen is down, the turtle also draws.
   2. right(degree): turn its face to right with a degree.

**Example:**

Consider [square.py.txt](https://dcm.uhcl.edu/yue/courses/csci1470/Fall2025/notes/turtle/square.py.txt) (remove .txt when download):

import turtle  
  
# Create a screen object in turtle  
screen = turtle.Screen()  
  
# Set screen dimensions  
screen.setup(width=300, height=300)  
  
# Set background color  
screen.bgcolor("lightyellow")  
  
# Create a turtle object  
pen = turtle.Turtle()  
pen.shape("turtle") # Change the turtle's shape  
pen.color("green") # Set the turtle's color  
pen.pensize(3) # Set the pen's size  
  
pen.forward(100) # Move forward by 100 units  
pen.right(90)  
pen.forward(100) # Move forward by 100 units  
pen.right(90)  
pen.forward(100) # Move forward by 100 units  
pen.right(90)  
pen.forward(100) # Move forward by 100 units  
  
screen.exitonclick()

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Current turtle location: (0,0) facing right (default initial setting of the turtle).

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

* Download and run [square2.py.txt](https://dcm.uhcl.edu/yue/courses/csci1470/Fall2025/notes/turtle/square2.py.txt). This is a version of square.py.txt that you can use as the basis of your HW assignment.
* Also, try to run it in IDLE by copying and pasting one statement at a time.

**Explanations:**

* "import turtle": import the built-in turtle module. This allows the program to use the turtle object.
* "pen = turtle.Turtle()" creates a turtle object and refers to it as "pen"
* "pen.forward(100)" moves the pen. Since the pen is down, the turtle also draws.
* "pen.right(90)" turns the turtle right 90 degrees.
* screen.exitonclick(): pause until the screen is clicked on to exit.