// Killing Cancer with Code
// Richard Popple

isCancerKilled = false;
while ( !isCancerKilled )
{
   isCancerKilled = cancer.Kill();
}

Why scripting?
• Automate planning tasks
• Automate QA
• Data mining

Script types
• Macros
• Read only
• Full read/write
Script types
- Macros
- Read only
- Full read/write

QA and safety
- Wild West
- No AAPM guidance
- Follow general standards of practice

Clinical scripts are the same as other clinical software
- Acceptance testing
- Commissioning
- Training and documentation
- Routine QA
- Version validation
  - New version of script
  - New version of host API/TPS
The Therac-25: every physicist's nightmare

Error messages provided to the operator were cryptic.

The operator's manual supplied with the machine does not explain nor even address the malfunction codes. The [Maintenance] Manual lists the various malfunction numbers but gives no explanation. The materials provided give no indication that these malfunctions could place a patient at risk.

A small amount of software testing was done on a simulator, but most testing was done on a system. It appears that unit and software testing was minimal, with most effort directed at the integrated system test. At a Therac-25 accident, the operator was not able to understand the messages being displayed.


DO NOT circumvent API restrictions

• Even if you are "smart" and "careful"

"Computer hacking refers to the practice of modifying or altering computer software and hardware to accomplish a goal that is considered to be outside of the creator's original objective."

"…computer hacking is somewhat ambiguous and difficult to define."

http://cyberlaws.comhacking

Basic principles
Basic principles

- Good variable naming
- Avoid global variables like poison
- Comments
- Short functions
- Documentation
- Tests

Cryptic

\[ g = 72.3; \]
Not much better

gAng = 72.3;

Getting there, but unnecessarily abbreviated

gant_ang = 72.3;

Hungarian notation – don’t do this

floatGantry = 72.3;
gantryAngle = 72.3;


Refactoring – encapsulates & improves readability

// Find the angle between gantry angles by taking the dot product.
// First convert from degrees to radians
double theta0 = 2 * Math.PI * gantryAngle[0] / 360.0;
double theta1 = 2 * Math.PI * gantryAngle[1] / 360.0;
// Compute the dot product
double dotProduct = Math.Cos(theta0) * Math.Cos(theta1);
dotProduct += Math.Sin(theta0) * Math.Sin(theta1);
// Compute the inverse cosine to get the angle and convert to degrees
double deltaAngle = 180.0 * Math.Acos(dotProduct) / Math.PI;

Refactoring – encapsulates & improves readability

double deltaAngle = GantryAngleDifferenceDeg(gantryAngle[0], gantryAngle[1]);
And now what you came for!