Chotipat Pornavalai
PSIT2016
Lecture 3

# Programming Concepts

- Use module "TurtleWorld", a module in package "swampy", to learn several important concepts in programming
  - Encapsulation
  - Generalization
  - Interface Design
  - Refactoring

- Others include docstring and debugging

- Download and Install
  - http://www.greenteapress.com/thinkpython/swampy/install.html

# Sample Code

```
from swampy.TurtleWorld import * # Python 2
from TurtleWorld import * # Python 3
world = TurtleWorld()
bob = Turtle()
alice = Turtle() # another turtle
bob.set_color(“blue”)
world.clear() # clear the world and turtles
bob.x = 0
bob.y = 0
bob. heading = 0 # heading to the east
bob.redraw()
bob.delay = 0.0001 # move faster
wait_for_user() # put this at the end of your code
```

# Encapsulation

- Draw a square

- Draw square (using for)

```
for i in range(4):
    fd(bob, 100)
    lt(bob)
```

- Bob and Alice need same code, why don't encapsulate to function?

- Wrapping a piece of code up in a function is called **encapsulation**

# Generalization

- Adding parameters to a function is called **generalization**
- Square for any size?
- Why only square? We need polygon
- It is legal and sometimes helpful to include the name of parameters in the argument list. These are called “**keyword arguments**””, ex. polygon(bob, n=7, length=70)
- More generalization? We need circle!

# Interface Design

- The interface of a function is a summary of how it is used: what are the parameters? What does the function do? And what is the return value?
- An interface is “clean” if it is “as simple as possible, but not simpler. (Einstein)”
- Circle is approximation of polygon of size ***n***
- Should ***n*** be one the circle interface?
- More generalization? We need arc!

# Refactoring

- Arc is similar to circle but we cannot reuse polygon as we did on circle

- Should we generalize polygon function by adding angle as its interface?

- We should generalize polygon but changing its name to polyline

- The process of rearranging a program to improve function interfaces and facilitate code reuse is “**refactoring**”

- After refactoring, circle is just a call of arc, and polygon is just a call of polyline

# A Development Plan

- A process for writing programs
- The process we use is "**encapsulation and generalization**"
- 5 Steps:
  - 1. write program with no function definition,
  - 2. once working,  encapsulate to a function and name it
  - 3. generalize function by adding appropriate arguments
  - 4. Repeat 1-3
  - 5. Look for opportunities to improve program by refactoring

# Docstring

- A string at the beginning of a function (under header and before its body) that explains the interface, also known as **mutiline string**

- It is important part of interface design, and well-designed interface should be simple to explain

- Example:

```
def polyline(t, length, n, angle):
    """Draw n line segments with the given length and angle (in
    degrees) between them. t is turtle.
    """
    for i in range(n):
        ……
```

# Debugging

- Interface is like a **contract** between a function and a caller.

- The caller agrees to provide certain parameters and the function agrees to do certain work.

- **Preconditions:** requirements on the interface (such as number of parameters,  the order , types, and semantics of parameters) must be met (true) before the function starts executing.

- **Postconditions:** the intended effect of the function

# Debugging

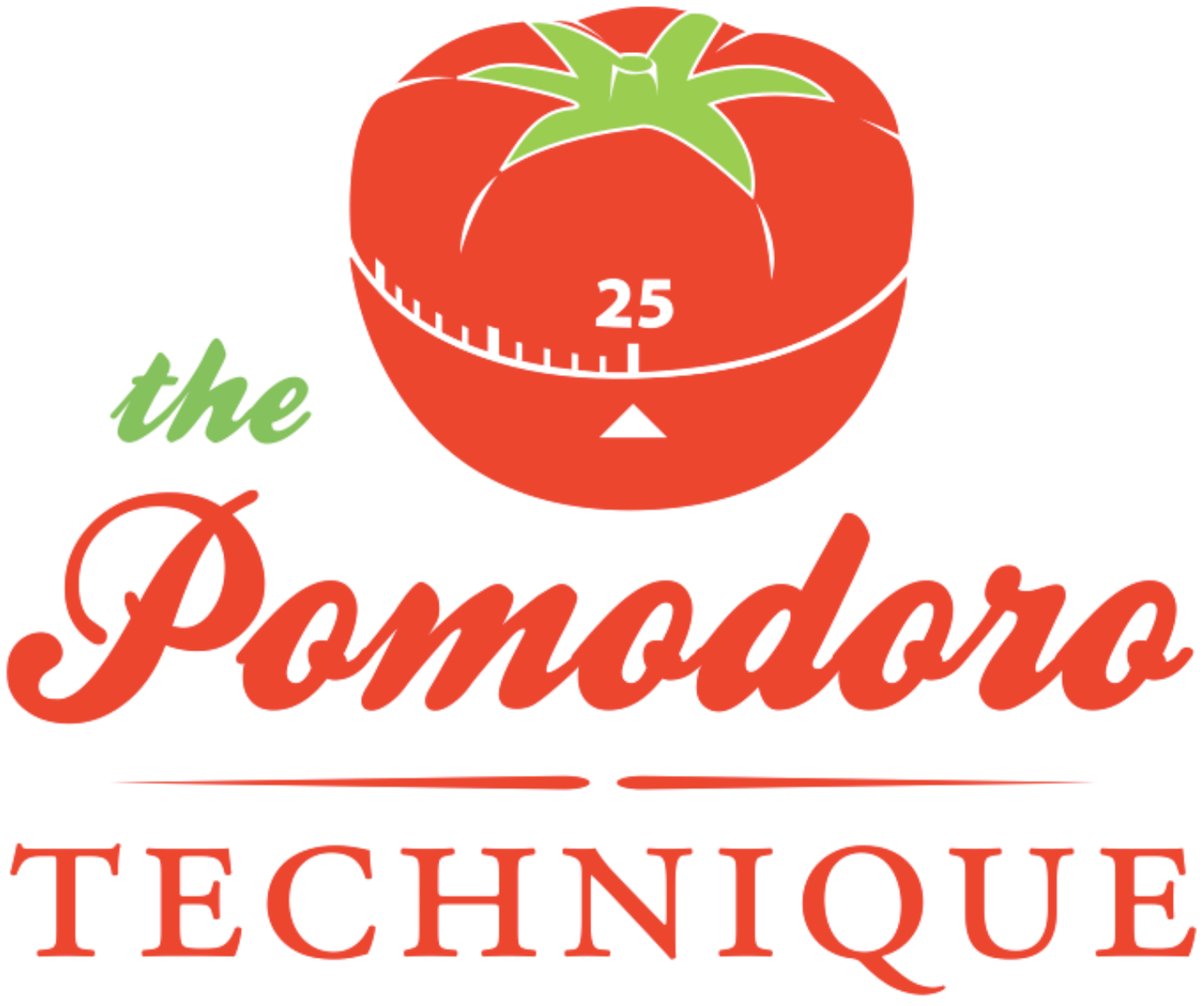
- Preconditions are the responsibility of the caller
- How and why the caller violates preconditions?
- It is a good idea for functions to check their preconditions rather than assume they are true, before starting

# po·mo·do·ro

/ˌpämәˈdôrō/

*adjective*

denoting a sauce made from tomatoes, typically served with pasta.
"a low-calorie pomodoro sauce is the way to go if you're planning on enjoying some pasta"




http://caps.ucsd.edu/Downloads/tx_forms/koch/pomodoro_handouts/ThePomodoroTechnique_v1-3.pdf

# The teddy bear principle in programming

It's very simple: Put a teddy bear on your desk. When you have a programming problem, explain what you are doing to the teddy bear. Eureka is a likely outcome of this method.

http://compaspascal.blogspot.com/2007/12/teddy-bear-principle-in-programming.html