# Week 5, Lecture 2

Dr. Bodine

# Last time

- Decisions
  - Conditions: expressions and relational operators
- If statements!!!
  - If, If-else, if-elif-else
- Boolean logic
  - Two states: True or False in python

# This time

- Continue with decisions
  - Nested decisions
  - How to approach decision making
- Exception handling
  - Protect against errors

# Decisions

- Simple decisions
  - if statements

```
if x<0: print("x is negative"
```

- Two-way decisions
  - if-else statements

```
if x<0: print("x is negative")
else: print()
```

- Multi-way decisions
  - if-elif-else statements

```
if x<0: print("x is negative")
elif x == 0: print("x is 0")
else: print("x is positive")
```

# Book question:

- Textbook Chapter 7, Question 6: The speeding ticket fine policy for Podunksville is $50 plus $5 for each 5 mph over the limit, plus a penalty of $200 for any speed over 90 mph. Write a program that accepts a speed limit and a clocked speed and either prints a message indicating the speed was legal or prints the amount of the fine, if the speed was illegal.
  - Can we write a mathematical function that would determine the fine?
  - Can we write it in terms of logical statements (if, if-else, if-elif)?
  - Can we implement it in a program?

# Book question: solution

```
>>> def speedtest(speedlimit,clockedspeed):
    if(speedlimit>=clockedspeed):
        print("The speed was within the legal speedlimit.")
    else:
        print("The speed was over the legal speedlimit.")
        fine = 50 + 5*((clockedspeed-speedlimit)//5)
        if clockedspeed > 90:
            fine += 200
        print("The fine will be $",fine)


>>> speedtest(70,99)
The speed was over the legal speedlimit.
The fine will be $ 275
>>> speedtest(25,21)
The speed was within the legal speedlimit.
>>>
```

# Nested Decisions

- If something is true, test if something else is also true

```
if <condition>:
     <some code to run>
     if <condition2>:
          <some other code to run
     else: <do this if condition 2 not met>
else: <do this if condition 1 not met>
```

# Nested Decisions

- If something is true, test if something else is also true

```
if <condition>:
     <some code to run>
     if <condition2>:
          <some other code to run
     else: <do this if condition 2 not met>
else: <do this if condition 1 not met>
```

- Previous solution used NESTED decision
  - Did it have to use this form? What happens if we do it without nested decisions?

# Nested Decisions

```
def speedtest(speedlimit,clockedspeed):
    if(speedlimit>=clockedspeed):
        print("The speed was within the legal speedlimit.")
    else:
        print("The speed was over the legal speedlimit.")
        fine = 50 + 5*((clockedspeed-speedlimit)//5)
        if clockedspeed > 90:
            fine += 200
        print("The fine will be $",fine)
```

```
def speedtest2(speedlimit,clockedspeed):
    if(speedlimit>=clockedspeed):
        print("The speed was within the legal speedlimit.")
    elif(clockspeed>90)
        print("The speed was over the legal speedlimit.")
        fine = 50 + 5*((clockedspeed-speedlimit)//5) +200
        print("The fine will be $",fine)
    else:
        print("The speed was over the legal speedlimit.")
        fine = 50 + 5*((clockedspeed-speedlimit)//5)
        print("The fine will be $",fine)
```

# Things to keep in mind about decisions

- When trying to define conditions, think carefully about how you would make the decision. We often employ implicit knowledge when we calculate things…make that explicit

- Unless the task is trivial, there are multiple ways to do it. Think through pros and cons before committing to a final design

- Consider a general problem to guide your thinking. Often there is an elegant general solution.

# Exception handling

- Graceful exit in cases where something doesn't go as planned
- Could use an explicit if statement but some languages have built-in ways to handle exceptions
- Can often use same code but add what to do in case of an error
- In python uses a "try-except" format

```
try:
    <what you want to do>
except <error type>:
    <what to do in case of error>
```

# Looking at errors

1. What types of input have we used so far?
2. What type of errors could be made in giving that input?
3. What type of errors result from those inputs?
4. How do we protect against those errors?

# Types of errors

- NameError
- TypeError
- SyntaxError
- ValueError
- ZeroDivisionError

- How do we know which errors to protect against?
- Do we guess?
- Do we try things?

# Example:

- Assignment 1: reciprocal and square

```
#test assignment1

def main():
    value = eval(input("Enter a number "))
    square = value**2
    reciprocal = 1/value
    print("The reciprocal of",value,"is",reciprocal)
    print("The square of",value,"is",square)



main()
```

- What happens if we enter something wrong?

# Example exception handling:

- What happens if we enter a **0** when prompted for a number?

```
Enter a number 0
Traceback (most recent call last):
  File "<pyshell#562>", line 1, in <module>
    test()
  File "<pyshell#561>", line 4, in test
    reciprocal = 1/value
ZeroDivisionError: division by zero
```

- Add a try-except statement to handle this error:

```
def test():

    try:
        value = eval(input("Enter a number "))
        square = value**2
        reciprocal = 1/value
        print("The reciprocal of",value,"is",reciprocal)
        print("The square of",value,"is",square)

    except ZeroDivisionError:
        print("\nInvalid input: reciprocal of 0 is undefined")
```

```
>>> test()
Enter a number 0

Invalid input: reciprocal of 0 is undefined
```

# Example exception handling:

- What happens if we enter a **nothing** when prompted for a number?

```
Enter a number
Traceback (most recent call last):
  File "<pyshell#580>", line 1, in <module>
    test()
  File "<pyshell#578>", line 3, in test
    value = eval(input("Enter a number "))
  File "<string>", line 0

    ^
SyntaxError: unexpected EOF while parsing
```

- Add an exception for this error

```
def test():

    try:
        value = eval(input("Enter a number "))
        square = value**2
        reciprocal = 1/value
        print("The reciprocal of",value,"is",reciprocal)
        print("The square of",value,"is",square)

    except ZeroDivisionError:
        print("\nInvalid input: reciprocal of 0 is undefined")
    except SyntaxError:
        print("\nInvalid input: you need to enter a value")
```

```
>>> test()
Enter a number

Invalid input: you need to enter a value
>>>
```

# Example exception handling:

- What happens if we enter a **string** when prompted for a number?

```
Enter a number f
Traceback (most recent call last):
  File "<pyshell#585>", line 1, in <module>
    test()
  File "<pyshell#583>", line 4, in test
    value = eval(input("Enter a number "))
  File "<string>", line 1, in <module>
NameError: name 'f' is not defined
```

- Add an exception for this error

```
def test():

    try:
        value = eval(input("Enter a number "))
        square = value**2
        reciprocal = 1/value
        print("The reciprocal of",value,"is",reciprocal)
        print("The square of",value,"is",square)

    except ZeroDivisionError:
        print("\nInvalid input: reciprocal of 0 is undefined")
    except SyntaxError:
        print("\nInvalid input: you need to enter a value")
    except NameError:
        print("\nInvalid input: please enter a number")
```

```
>>> test()
Enter a number f

Invalid input: please enter a number
```

# Example exception handling:

- What happens if we enter a **string literal** when prompted for a number?

```
Enter a number '5'
Traceback (most recent call last):
  File "<pyshell#597>", line 1, in <module>
    test()
  File "<pyshell#596>", line 5, in test
    square = value**2
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

- Add an exception for this error

```
def test():

    try:
        value = eval(input("Enter a number "))
        square = value**2
        reciprocal = 1/value
        print("The reciprocal of",value,"is",reciprocal)
        print("The square of",value,"is",square)

    except ZeroDivisionError:
        print("\nInvalid input: reciprocal of 0 is undefined")
    except SyntaxError:
        print("\nInvalid input: you need to enter a value")
    except NameError:
        print("\nInvalid input: please enter a number")
    except TypeError:
        print("\nInvalid input: please enter a number")
```

# Example exception handling:

- Can add a general exception for when anything else goes wrong.

```
def test():

    try:
        value = eval(input("Enter a number "))
        square = value**2
        reciprocal = 1/value
        print("The reciprocal of",value,"is",reciprocal)
        print("The square of",value,"is",square)

    except ZeroDivisionError:
        print("\nInvalid input: reciprocal of 0 is undefined")
    except SyntaxError:
        print("\nInvalid input: you need to enter a value")
    except NameError:
        print("\nInvalid input: please enter a number")
    except TypeError:
        print("\nInvalid input: please enter a number")
    except:
        print("\nUnspecified error: please try again.")
```

# Think-pair-share

- Can we always protect against all input errors?
- How do we know what exceptions to add to our script?
- Do all programs need exception handling?

# Review

- Definition of computer and computer science
- Basic python programs
- Input/Output
- Numeric types
- Basic loops
- String type
- Files
- Functions
- Decisions

# Computers and computer science

- Computer
  - Machine that stores and manipulates information based on a changeable program
- Parts of a computer
  - CPU
  - RAM
  - Secondary memory
  - Keyboard?
  - Monitor?
- Computer science
  - The study of what is computable
  - Just hardware?
  - Just software?

# Basic python programs

```
#This is a basic python program

def main():
    print("Hello, World!")

main()
```

# Input/Output

- Input statements
  - Prompt user for something

- Output print statements
  - How to include text
  - How to include value of variables
  - White space

# Numeric types

- Int
  - Whole numbers
- Float
  - Numbers with a decimal portion (including 3.0)
- Arithmetic: +,-,*,/,//,%, abs
- Math library
  - Import!

# Basic loops

```
for <index> in <something>:
    <do something
```

- Loop over elements in something
  - Entries in a list
  - Characters in a string
  - range(<integer>)
  - range(len(<mylist>))

# Strings

- String literals
  - ‘test’ vs “test”
- Strings as sequences of characters
  - Indexing
  - Slicing
  - Concatenating
- Splitting!
  - Whitespace, \n, or comma
- Immutable
  - Cannot assign based on index
  - Why not?

# Files

- Opening
  - Associates file with an object in program
- Reading
  - read()
  - readline()
  - readlines()
- Closing
  - Cleaning up memory

# Functions

- Subprograms
  - Standalone bits of code
- Inputs
  - Parameters
  - What the function has access to while running
- Definition of function
  - Formal parameters
- Call to the function
  - Actual parameters

# Functions part 2

- Passing parameters
  - By value vs be reference
  - Lists vs strings
- Outputs
  - Void vs return-valued
  - Return statements
- SCOPE!
  - Where are variables defined

# Decision

- Basic Boolean decisions
  - True or False
- If statements
- If-else
- If-elif-else
- Nested