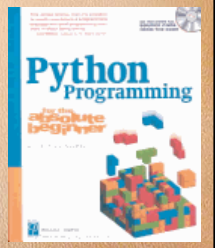
# Python Programming: An Introduction to Computer Science

Chapter 2

Dan Fleck






## The Software Development Process

- The process of creating a program is often broken down into stages according to the information that is produced in each phase.

1. Analyze the problem
2. Determine requirements
3. Create a Design
4. Implementation
5. Testing
6. Maintenance



## The Software Development Process

- **Analyze the Problem**
  Figure out exactly the problem to be solved. Try to understand it as much as possible.



## The Software Development Process

- **Determine Requirements**
  Describe exactly what your program will do.
  - Don't worry about ***how*** the program will work, but ***what*** it will do.
  - Includes describing the inputs, outputs, and how they relate to one another.

## The Software Development Process

- **Create a Design**
  - Formulate the overall structure of the program.
  - This is where the ***how*** of the program gets worked out.
  - You choose or develop your own algorithm that meets the requirements.



## The Software Development Process

- **Implement the Design**
  - Translate the design into a computer language.
  - In this course we will use Python.

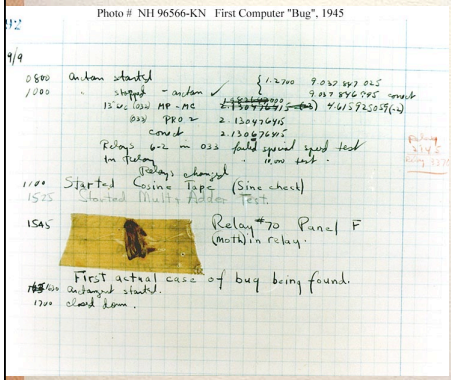


## The Software Development Process

- **Test/Debug the Program**
  - Try out your program to see if it worked.
  - If there are any errors (*bugs*), they need to be located and fixed. This process is called *debugging*.
  - Your goal is to find errors, so try everything that might "break" your program! (Correct and incorrect inputs)



## Why is it called debugging?




**The First "Computer Bug"**
Moth found trapped between points at Relay # 70, Panel F, of the Mark II Aiken Relay Calculator while it was being tested at Harvard University, 9 September 1945. The operators affixed the moth to the computer log, with the entry: **"First actual case of bug being found"**. They put out the word that they had **"debugged"** the machine, thus introducing the term "debugging a computer program".

Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988.*U.S. Naval Historical Center Photograph.*

## The Software Development Process

- **Maintain the Program**
  - Continue developing the program in response to the needs of your users.
  - In the real world, most programs are never completely finished – they evolve over time.



## Example : Temperature Converter Analysis

- Analysis – the temperature is given in Celsius, user wants it expressed in degrees Fahrenheit.
- Requirements
  - Input – temperature in Celsius
  - Output – temperature in Fahrenheit
  - Output = 9/5(input) + 32



## Example : Temperature Converter Design

- Design
  - *Input*: Prompt the user for input (Celsius temperature)
  - *Process*: Process it to convert it to Fahrenheit using F = 9/5(C) + 32
  - *Output:* Output the result by displaying it on the screen



## Example : Temperature Converter

- Before we start coding, let's write a rough draft of the program in *pseudocode*
- Pseudocode is precise English that describes what a program does, step by step. However, There is no "official" syntax for pseudocode
- Using pseudocode, we can concentrate on the algorithm rather than the programming language.

## Temperature Converter Pseudocode

- Pseudocode:
  - Input the temperature in degrees Celsius (call it celsius)
  - Calculate fahrenheit as (9/5)*celsius+32
  - Output fahrenheit
- Now we need to convert this to Python!



## Temperature Converter Python Code

```
#convert.py
# A program to convert Celsius temps to Fahrenheit
# by: Susan Computewell

def main():
    celsiusString = raw_input("What is the Celsius temperature? ")
    celsius = int(celsiusString) # Convert from a string to an integer (number)
    fahrenheit = (9.0/5.0) * celsius + 32
    print "The temperature is ",fahrenheit," degrees Fahrenheit."

main()
```

Lets try it in IDLE after the next slide



## Using IDLE a Python Development Environment

- Open IDLE
- In the Python shell you can run dynamic Python commands (this shell is the window that opens)
- File→ New opens the window to write a program
- Run→ Run Module runs your program (or press F5)



## How to run outside of IDLE

- If you have a python source file (something.py) to run it outside of IDLE on the command line:
  - C:\python something.py
  - C:\python c:\SomeDir\myPythonFiles\something.py
  - **WARNING:** This Python is on your PATH. To add it see this video: http://showmedo.com/videos/video?name=960000
  - On Mac this is typically done automatically.

## Question

- Does that mean I can create a Python source file in anything, not just in IDLE? Like Windows Notepad? Or something else?
- Answer: Yes! IDLE is an integrated development environment (IDE), so it makes it EASIER, but you can use any plain-text editor. (MS Word isn't plain text.)



## Need more IDLE Help?

- Try reading this webpage on using IDLE:
- http://hkn.eecs.berkeley.edu/~dyoo/python/idle_intro/index.html



## Temperature Converter Testing

- Once we write a program, we should test it!
- What are some values with known answers?

```
>>>
What is the Celsius temperature? 0
The temperature is  32.0  degrees Fahrenheit.
>>> main()
What is the Celsius temperature? 100
The temperature is  212.0  degrees Fahrenheit.
>>> main()
What is the Celsius temperature? -40
The temperature is  -40.0  degrees Fahrenheit.
>>>
```



## Program Revisited

```
#convert.py
# A program to convert Celsius temps to Fahrenheit
# by: Susan Computewell

def main():
    celsiusString = raw_input("What is the Celsius temperature? ")
    celsius = int(celsiusString) # Convert from a string to an integer
     (number)
    fahrenheit = (9.0/5.0) * celsius + 32
    print "The temperature is ",fahrenheit," degrees Fahrenheit."

main()
```

Comments

starts a function definition

## Elements of Programs : Identifiers

- Names of variables: celsius, fahrenheit
- Names of functions: range, main, input
- Names of modules: convert

These names are called *identifiers*

- Every identifier must begin with a letter or underscore ("_"), followed by any sequence of letters, digits, or underscores.
- Good programmers use **meaningful names**
- Identifiers are case sensitive.



## Elements of Programs : Identifiers

Identifiers are case sensitive.

- In Python, identifiers:
  - myVar
  - MYVAR
  - myvar
- Are all *DIFFERENT* because Python is case-sensitive

Lets try it in IDLE



## Reserved Words

Some identifiers are part of Python itself. These identifiers are known as *reserved words*. This means they are not available for you to use as a name for a variable, etc. in your program.

| | | | | |
|---|---|---|---|---|
| and | del | for | is | raise |
| assert | elif | from | lambda | return |
| break | else | global | not | try |
| class | except | if | or | while |
| continue | exec | import | pass | yield |
| def | finally | in | print | |

Table 2.1: Python Reserved Words.



## Using identifiers in expressions

```
>>> x = 5
>>> x
5
>>> print x
5
>>> print spam

Traceback (most recent call last):
  File "<pyshell#15>", line 1, in -toplevel-
    print spam
NameError: name 'spam' is not defined
>>>
```

- NameError is the error when you try to use a variable without a value assigned to it.

## Math Operators

- Simpler expressions can be combined using *operators*.
- +, -, *, /, **, %
- Spaces are irrelevant within an expression.
- The normal mathematical precedence applies.
- ((x1 – x2) / 2*n) + (spam / k**3)

Precedence is:
PEMDAS - (), **, *, /, +, -

```
>>>
>>>
>>> x = 4
>>> y = 2
>>> x + y
6
>>> x - y
2
>>> x * y
8
>>> x / y
2
>>> x ** y
16
>>>
>>> x * 3
12
>>> d = x * 3
>>> d
12
>>>
```



## Elements of Programs

- Output Statements
  - A print statement can print any number of expressions.
  - Successive print statements will display on separate lines.
  - A bare print will print a blank line.
  - If a print statement ends with a ",", the cursor is not advanced to the next line.



## Elements of Programs

```
print 3+4
print 3, 4, 3+4
print
print 3, 4,
print 3+      4
print "The answer is",
   3+4
```

```
>>>
>>>
>>>
>>> print 3 + 4
7
>>> print 3, 4, 3+4
3 4 7
>>> print

>>> print 3,4 ; print 3+   4
3 4
7
>>> print 3,4, ; print 3+   4
3 4 7
>>> print "The answer is", 3+4
The answer is 7
>>>
```



## Assignment Statements

- <variable> = <expr>
  variable is an identifier, expr is an expression
- The expression on the RHS is evaluated to produce a value which is then associated with the variable named on the LHS.
- x = 3.9 * x * (1-x)
- fahrenheit = 9.0/5.0 * celsius + 32
- x = 5

## Assignment Statements

- Variables can be reassigned as many times as you want!

```
>>> myVar = 0
>>> myVar
0
>>> myVar = 7
>>> myVar
7
>>> myVar = myVar + 1
>>> myVar
8
>>>
```



## Assigning Input

- Input: gets input from the user and stores it into a variable.
- <variable> = raw_input(<prompt>)
- The raw_input function ALWAYS returns a String (but you can convert it to a number)

```
>>> x = raw_input("Give me a number >")
Give me a number >35
>>> type(x)
<type 'str'>
>>> y = int(x)
>>> type(y)
<type 'int'>
>>>
```



## Converting Strings to Numbers

- `someFloatString = '1.343'`
- `someVar = float(someFloatString)`
- `print someVar`
- More on this later…

| Function | Meaning |
|---|---|
| float(<expr>) | Convert expr to a floating point value |
| int(<expr>) | Convert expr to an integer value |
| long(<expr>) | Convert expr to a long integer value |
| str(<expr>) | Return a string representation of expr |
| eval(<string>) | Evaluate string as an expression |



## Assigning Input

- First the prompt is evaluated
- The program waits for the user to enter a value and press <enter>
- The expression that was entered is assigned to the input variable as a string.

```
>>> def inp():
        yourInput = raw_input('Input something >')
        print 'You entered:',yourInput


>>> inp() # Call the function
Input something >Barack Obama won
You entered: Barack Obama won
>>>
```

## How to find information yourself

- To the Python docs!
  - http://docs.python.org/
- Library Reference – kinda hard to read
  - http://docs.python.org/library/index.html
- Language Reference – very easy to read
  - http://docs.python.org/reference/index.html




## Strings: Can there be more?

- Lets try and see what else we can find about them in the Python Documentation.
- What Methods are supported?
  - How can find if a string is all numbers?
  - How can I find the index of character "h" in "Python"?



## String Methods you should know

- upper
- lower
- replace
- count
- find
- isdigit
- split

Use the dot operator to call a function that is part of a class:

```
firstName = "Dan"
yellName = firstName.upper()
```



## Using non-built in modules

- The math module adds more functions to use like cosine and sine, square root, etc…
- Lets try it!
- Math is NOT built-in (like String) so we need to tell Python we want to use it

## Using Python's Modules

- Python has a lot of code available in modules for you to use
- Using modules, you must "import" them.

```
pythag.py – /Users/dfleck/Documents/gmu
# Example to calculate the pythagorean theorm
# a^2 + b^2 = c^2

import math

def pythag():
    a = input("What is a >")
    b = input("What is b >")
    c = math.sqrt(math.pow(a, 2) + math.pow(b, 2))
    return c

# Run the function
temp = pythag()
print "The length of side c is:", temp
```

```
>>> ========================
>>>
What is a >3
What is b >4
The length of side c is: 5.0
>>>
```



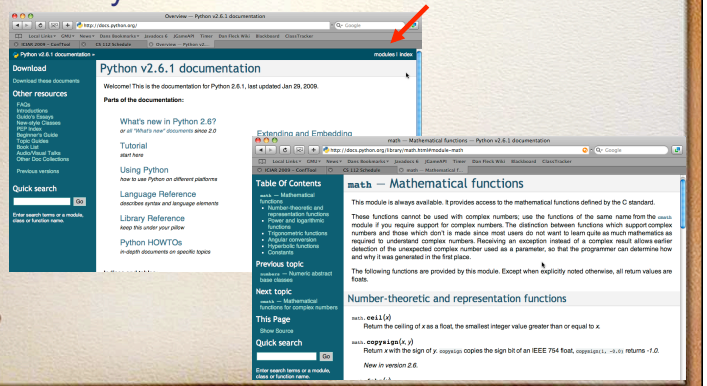
## Import Statement

- Importing a module tells the Python interpreter you plan to use some or all of the function definted in that module.
- To use those functions though, you must prepend the name of the module:
- import math
- Then call function XYZ defined in the math module as:
  - someVar = math.XYZ()



## What modules are available?

- Many! Find info in the module index






## String Concatenation and Multiplication

- pStr = "Python"
- rStr = "Rocks"
- prStr = pStr + rStr
- This is string concatenation – adding two strings to get a new string

- pMulti = pStr * 5  # Use variable
- pMulti = "Hello" * 3  # Use a literal value
- This is string "multiplication", says create a new string by concatenating 5 in a row

## Another way to print – Fill in the blank

Placeholder for ints

```
>>> numCats=5
>>> numDogs=7
>>> print "There were %d cats and %d dogs" %(numCats, numDogs)
There were 5 cats and 7 dogs
>>>
```

Printing is the SAME as creating a new string. You are really formatting the String and then printing it. This also works:

```
>>> myString = "There were %d cats and %d dogs" %(numCats, numDogs)
>>> print myString
There were 5 cats and 7 dogs
```

Inside the String you can put placeholders for other values. The placeholders specify a type.

- %d = Signed integer
- %f = Floating point (decimal format)
- %s = String



## String formats

You can also specify a minimum field width like this: "%20d" . This will force the number to take up 20 spaces.

```
>>> print "Num 1 = %10f" %(123.456)
Num 1 = 123.456000
```

To print in columns:

```
print "Col1       Col2"
print "%10f       %10f" %(12.23, 222.45)
print "%10f             %10f" %(444.55, 777)
Col1           Col2
 12.230000     222.450000
444.550000     777.000000
```



## String formats

Lots of other String formats are found here:

http://docs.python.org/library/stdtypes.html#string-formatting



## A few final words on Math

- Shortcut operators are available:
  - x = x + 1
  - x += 1
  - someVariable = 13 * someVariable
  - someVariable *= 13
  - *= /= += -= %=
  - Do the operation on the current variable, and save the result.

## Number Bases

- Decimal: Digits 0-9
- Binary: Digits 0-1
- Hexadecimal: Digit 0-9, A-F

$23 = (2*10^{1})+(3*10^{0})$ # Decimal (base 10)

$3001 = (3*10^{3}) + (0*10^{2}) + (0*10^{1})+ (1*10^{0})$

In base 10 the red digits can be what?



## Binary: Base 2

$23 = 2*10^{1}+3*10^{0}$ # Decimal (base 10)

- $2^6 = 64$
- $2^5 = 32$
- $2^4 = 16$
- $2^3 = 8$
- $2^2 = 4$
- $2^1 = 2$
- $2^0 = 1$

$23 = 16 + 4 + 2 + 1$
$= (1*2^4) + (0*2^3) + (1*2^2) + (1*2^1) + (1*2^0)$
$= 0b10111$

48 =

0b101 = =

0b1101 = =



## Binary: Base 2

$23 = 2*10^{1}+3*10^{0}$ # Decimal (base 10)

- $2^6 = 64$
- $2^5 = 32$
- $2^4 = 16$
- $2^3 = 8$
- $2^2 = 4$
- $2^1 = 2$
- $2^0 = 1$

$23 = 16 + 4 + 2 + 1$
$= (1*2^4) + (0*2^3) + (1*2^2) + (1*2^1) + (1*2^0)$
$= 0b10111$

$48 = (1*2^5) + (1*2^4) + (0*2^3) + (0*2^2) + (0*2^1) + (0*2^0)$
$= 0b11000$

$0b101 = (1*2^2) + (0*2^1) + (1*2^0) = 5$

$0b1101 = (1*2^3) + (1*2^2) + (0*2^1) + (1*2^0) = 13$



## Hexadecimal: Base 16

Digits: 0-9, A-F: A=10, B=11, C=12,.. F=15

- $16^3 = 4096$
- $16^2 = 256$
- $16^1 = 16$
- $16^0 = 1$

$23 = 16 + 4$
$= (1*16^1) + (4*16^0) = 0x14$

48 = =

250 = =

163 = =

## Hexadecimal: Base 16

Digits: 0-9, A-F: A=10, B=11, C=12,.. F=15

- $16^3 = 4096$
- $16^2 = 256$
- $16^1 = 16$
- $16^0 = 1$

23 = 16 + 4
= $(1*16^1) + (4*16^0)$ = 0x14

48 = $(3*16^1) + (0*16^0)$ = 0x30

250 = $(15*16^1) + (10*16^0)$ = 0xFA

163 = $(10*16^1) + (3*16^0)$ = 0xA3



## General Approach: Decimal to Hex

- From Base 10 to Base 16
  1. Divide the decimal number by 16.
  2. Treat the division as an integer division.
  3. Write down the remainder (in hexadecimal).
  4. Divide the result again by 16.
  5. Treat the division as an integer division.
  6. Repeat step 2 and 3 until result is 0.
  7. The hex value is the digit sequence of the remainders from the last to first.

From here: http://www.permadi.com/tutorial/numDecToHex/



## General Approach: Decimal to Binary

- Same as last slide, but replace 16 with 2.



## In Python: Output Strings

- You need to display numbers as binary or hexadecimal strings:

- x = 32
- xHex = hex(32) #What data type is xHex?
- xBin = bin(32) # Only in Python 2.6+

## In Python: Input From User

- You need to input different bases. Python knows how to make a number from the string using "eval".

- x = eval('0x10') # What is x? What data type?
- x = eval('0b10')
- x = eval('10')

How do I ask a user for a String?

Coming up: It's all about the jokes 53

## It's all about the jokes

- There are 10 kinds of people in this world, those who know binary and those who don't.

- (I'll be here all week.)

End of presentation 54