Getting Started With Python Programming

- Tutorial: creating computer programs
- Variables and constants
- Input and output
- Operators
- Common programming errors
- Formatted output
- Programming style

Python

- This is the name of the programming language that will be used to illustrate different programming concepts this semester:
  - My examples will be written in Python
  - Your assignments will be written in Python
- Some advantages:
  - Free
  - Powerful
  - Widely used (Google, NASA, Yahoo, Activision, Electronic Arts etc.)
- Named after a British comedy

Monty Python © Monty Python

- Official website (Python the programming language, not the Monty Python): http://www.python.org

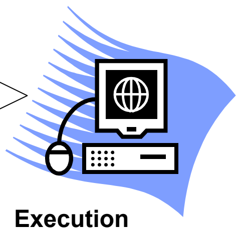
Online Help

- Getting Python (get version 2.X and not version 3.X)
  - http://www.python.org/download/
- Example of setting up Python on your computer (it's not mandatory to install Python at home, follow these instructions carefully, missteps occur at your own peril!)
  - http://docs.python.org/using/windows.html
- Explanation of concepts (along with examples to illustrate)
  - http://docs.python.org/tutorial/
- General documentation:
  - http://www.python.org/doc/

The Process Of Creating A Computer Program

Translation

- A special computer program (translator) translates the program that was just written by the programmer into the *only* form that the computer can understand (machine language)



Program Creation

- A person (programmer) writes a computer program (series of instructions).
- The program is written and saved using a text editor.
- The instructions in the programming language are high level (look much like a human language).

Execution

- The machine language instructions can now be directly executed by the computer.

An Example Python Program

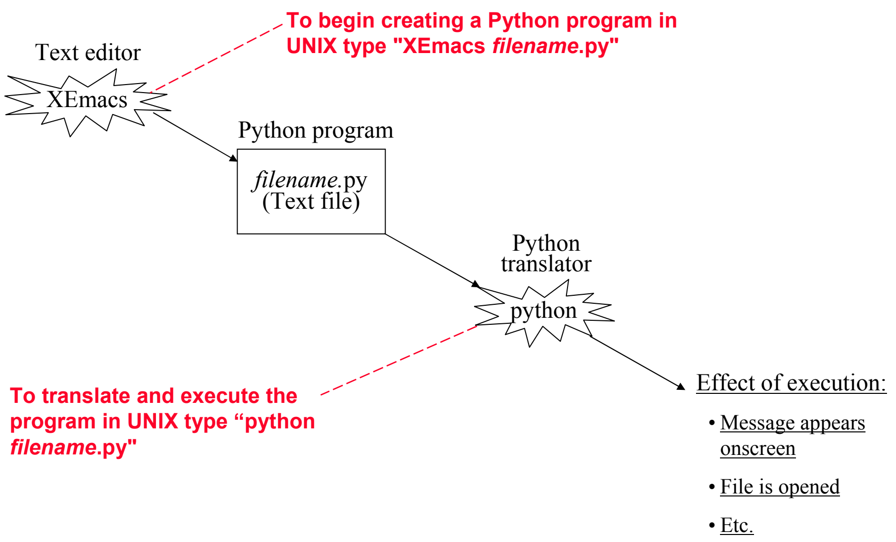
- You can find an online version of this program on the course web page and it is called "small.py"

Filename: small.py

```
print "hello"
```

James Tam

Creating, Translating And Executing Python Programs (CPSC Network[1])


To begin creating a Python program in UNIX type "XEmacs *filename*.py"
Text editor
XEmacs
Python program
filename.py (Text file)
Python translator
python
To translate and execute the program in UNIX type "python *filename*.py"
Effect of execution:
• Message appears onscreen
• File is opened
• Etc.

1 The CPSC (Computer Science) network runs on the UNIX operating system.

James Tam

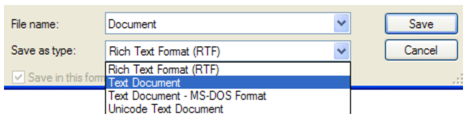
Creating, Translating And Executing The Sample Program (CPSC Network)

- **Creating the program in an editor**: Type “XEmacs small.py”
  - A file called “small.py” will be created in your UNIX account.

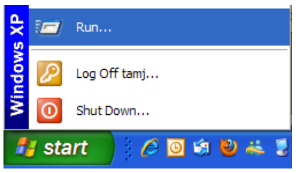
- **Translating and running the program**: Type “python small.py”
  - Make sure you type this command in the location where the Python program (small.py) is located.

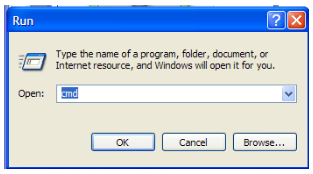
Creating Programs: Other Operating Systems

- The process is similar:
  - You need a text editor (e.g., WordPad, NotePad) to enter the program.
  - It can be done using any editor that you, want but don’t use a word processor (e.g., MS-Word) and remember to save it as a text file.



  - Also you need to open a command line to translate/run your Python program.





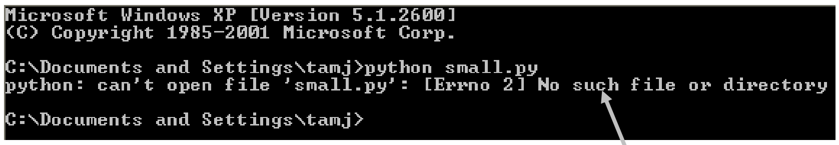
Run

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: cmd

Creating Programs: Other Operating Systems (2)

•When you translate/run your program in the command window make sure that you are in the same location as your Python program.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\tamj>python small.py
python: can't open file 'small.py': [Errno 2] No such file or directory

C:\Documents and Settings\tamj>
```

The Python program is in another folder.

Displaying String Output

•String output: A message appears onscreen that consists of a series of text characters.

•Whatever is contained with the quotes (single or double) is what appears onscreen.

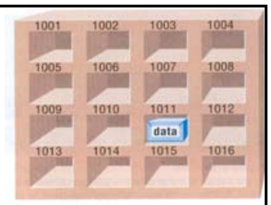
•**Format:**

print "*the message that you wish to appear*"

•**Example:**

```
print "foo"
print "bar"
```

Variables



- Set aside a location in memory.
- Used to store information (temporary).
  - This location can store one 'piece' of information.
  - *At most* the information will be accessible as long as the program runs.
- Some of the types of information which can be stored in variables:

  Format:
  <name of variable> = *<type of information to be stored in the variable>*

  Examples:
  - Integer (e.g., num = 10)
  - Real numbers (e.g., num = 10.0)
  - Strings (e.g., name = "james")

Picture from Computers in your future by Pfaffenberger B

Variable Naming Conventions

- Style requirement: The name should be meaningful.
- Style and Python requirement: Names *must* start with a letter (Python requirement) and *should not* begin with an underscore (style requirement).
- Python requirement: Can't be a keyword (see next slide).
- Style requirement: Names are case sensitive but avoid distinguishing variable names only by case.
- Style requirement: Variable names should generally be all lower case.
- Style requirement: For variable names composed of multiple words separate each word by capitalizing the first letter of each word (save for the first word) or by using an underscore. (Either approach is acceptable but be consistent!)

Key Words In Python[1]

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

1 From "*Starting out with Python*" by Tony Gaddis

Named Constants

- They are similar to variables: a memory location that's been given a name.
- Unlike variables their content *shouldn't* change.
- The naming conventions for choosing variable names generally apply to constants but the name of constants should be all UPPER CASE. (You can separate multiple words with an underscore).
- They are capitalized so the reader of the program can distinguish them from variables.
  - For some programming languages the translator will enforce the unchanging nature of the constant.
  - For languages such as Python it is up to the programmer to recognize a constant for what it is and not to change it.

Terminology: Named Constants Vs. Literals

- Named constant: given an explicit name
  - TAX_RATE = 0.2
  - afterTax = income – (income * TAX_RATE)

- Literal: not given a name, the value that you seen is literally the value that you have.
  - afterTax = 100000 – (100000 * 0.2)

Terminology: Named Constants Vs. Literals

- Named constant: given an explicit name
  - TAX_RATE = 0.2
  - afterTax = income – (income * TAX_RATE)

Named constants

- Literal: not given a name, the value that you seen is literally the value that you have.
  - afterTax = 100000 – (100000 * 0.2)

Literals

Why Use Constants

1. They make your program easier to read and understand

```
populationChange = (0.1758 – 0.1257) * currentPopulation;
```

Vs.

```
BIRTH_RATE = 17.58
MORTALITY_RATE = 0.1257
currentPopulation = 1000000
populationChange = (BIRTH_RATE - MORTALITY_RATE) *
    currentPopulation
```

In this case the literals are Magic Numbers (avoid whenever possible!)

Purpose Of Named Constants (2)

•2) Makes the program easier to maintain

- If the constant is referred to several times throughout the program, changing the value of the constant once will change it throughout the program.

Purpose Of Named Constants (3)

```
BIRTH_RATE = 0.1758
MORTALITY_RATE = 0.1257
populationChange = 0
currentPopulation = 1000000
populationChange = (BIRTH_RATE - MORTALITY_RATE) * currentPopulation
if (populationChange > 0):
    print "Increase"
    print "Birth rate:", BIRTH_RATE, " Mortality rate:", MORTALITY_RATE, " Population
  change:", populationChange
elif (populationChange < 0):
    print "Decrease"
    print "Birth rate:", BIRTH_RATE, " Mortality rate:", MORTALITY_RATE,  "Population
  change:", populationChange
else:
    print "No change"
    print "Birth rate:", BIRTH_RATE, " Mortality rate:", MORTALITY_RATE,  "Population
  change:", populationChange
```

Purpose Of Named Constants (3)

One change in the initialization of the constant changes every reference to that constant

```
BIRTH_RATE = 0.8
MORTALITY_RATE = 0.1257
populationChange = 0
currentPopulation = 1000000
populationChange = (BIRTH_RATE - MORTALITY_RATE) * currentPopulation
if (populationChange > 0):
    print "Increase"
    print "Birth rate:", BIRTH_RATE, " Mortality rate:", MORTALITY_RATE, " Population
  change:", populationChange
elif (populationChange < 0):
    print "Decrease"
    print "Birth rate:", BIRTH_RATE, " Mortality rate:", MORTALITY_RATE,  "Population
  change:", populationChange
else:
    print "No change"
    print "Birth rate:", BIRTH_RATE, " Mortality rate:", MORTALITY_RATE,  "Population
  change:", populationChange
```

Purpose Of Named Constants (4)

One change in the initialization of the constant changes every reference to that constant

```
BIRTH_RATE = 0.1758
MORTALITY_RATE = 0.01
populationChange = 0
currentPopulation = 1000000
populationChange = (BIRTH_RATE - MORTALITY_RATE) * currentPopulation
if (populationChange > 0):
    print "Increase"
    print "Birth rate:", BIRTH_RATE, " Mortality rate:", MORTALITY_RATE, " Population
  change:", populationChange
elif (populationChange < 0):
    print "Decrease"
    print "Birth rate:", BIRTH_RATE, " Mortality rate:", MORTALITY_RATE,  "Population
  change:", populationChange
else:
    print "No change"
    print "Birth rate:", BIRTH_RATE, " Mortality rate:", MORTALITY_RATE,  "Population
  change:", populationChange
```

Named Constants And Style

- Self documenting programs: it’s faster for the reader of a program to determine how the program works if the programmer writes the program in a clear and explicit fashion.
- (This means that there isn’t a need to explain how a program does what it does by written descriptions – called “documentation” – which we’ll talk about later).
- One example of writing a self-documenting program is by the use of named constants.

Reminder: Named Constants And Python

- Using named constants is regarded as "good" style when writing a computer program.
- Some programming languages have a mechanism for ensuring that named constants do not change.
- Example:

```
MY_CONSTANT = 100
MY_CONSTANT = 12
```

$\}$ **With programming languages that enforce the rule that constants can't change this would result in an error**

- Reminder: Python does not enforce the unchanging nature of constants so it is up to the person writing/modifying the program to avoid changing the value stored in a constant.

Displaying The Contents Of Variables And Constants

- **Format:**

print <*variable name*>
print <*constant name*>

- **Example (the online program is called 'output1.py'):**

```
aNum = 10
A_CONSTANT = 10
print aNum
print A_CONSTANT
```

Mixed Output

•Mixed output: getting string output and the contents of variables (or constants) to appear together.

•**Format:**

print "*string*", <*variable or constant*>, "*string*", <*variable or constant*> etc.

•**Examples (the online program is called 'output2.py'):**

```
myInteger = 10
myReal = 10.5
myString = "hello"

print "MyInteger:" , myInteger
print "MyReal:" , myReal
print "MyString:" , myString
```

$\}$ **The comma signals to the translator that the string and the contents of the variable should appear on the same line.**

Arithmetic Operators

Operator	Description	Example
=	Assignment	num = 7
+	Addition	num = 2 + 2
-	Subtraction	num = 6 - 4
*	Multiplication	num = 5 * 4
/	Division	num = 25 / 5
%	Modulo	num = 8 % 3
**	Exponent	num = 9 ** 2

Augmented Assignment Operators (Shortcuts)

Operator	Long example	Augmented Shortcut
+=	num = num + 1	num += 1
-=	num = num – 1	num -= 1
*=	num = num * 2	num *= 2
/=	num = num / 2	num /= 2
%=	num = num % 2	num %= 2
**=	num = num ** 2	num **= 2

Order Of Operation

•First level of precedence: top to bottom

•Second level of precedence

- If there are multiple operations that are on the same level then precedence goes from left to right.

()	Brackets (inner before outer)
**	Exponent
*, /, %	Multiplication, division, modulo
+, -	Addition, subtraction

Order Of Operation And Style

- Even for languages where there are clear rules of precedence (e.g., Java, Python) it is regarded as good style to explicitly bracket your operations.

```
x = (a * b) + (c / d)
```

- It not only makes it easier to read complex formulas but also a good habit for languages where precedence is not always clear (e.g., C++, C).

Program Documentation

- Program documentation: Used to provide information about a computer program to another *programmer* (writes or modifies the program).
- This is different from a user manual which is written for people who will *use the program*.
- Documentation is written inside the same file as the computer program (when you see the computer program you can see the documentation).
- The purpose is to help other programmers understand the program: what the different parts of the program do, what are some of it's limitations etc.

Program Documentation (2)

- It doesn't get translated into machine language.
- It doesn't contain instructions for the computer to execute.
- It's information for the reader of the program:
  - What does the program as a while do e.g., tax program.
  - What are the specific features of the program e.g., it calculates personal or small business tax.
  - What are it's limitations e.g., it only follows Canadian tax laws and cannot be used in the US. In Canada it doesn't calculate taxes for organizations with yearly gross earnings over $1 billion.
  - What is the version of the program
    - If you don't use numbers for the different versions of your program then consider using dates (tie versions with program features).

Program Documentation (3)

- **Format:**

$\underbrace{\text{\# <}\textit{Documentation}\text{>}}$

The number sign '#' flags the translator that what's on this line is documentation.

- **Examples:**

```
# Tax-It v1.0: This program will electronically calculate your tax return.
# This program will only allow you to complete a Canadian tax return.
```

Types Of Documentation

- Header documentation
- Inline documentation

Header Documentation

- Provided at the beginning of the program.
- It describes in a high-level fashion the features of the program as a whole (major features without a great deal of detail).

```
# HEADER DOCUMENTATION
# Word Processor features: print, save, spell check, insert images etc.

<program statement>
<program statement>
```

Inline Documentation

- Provided throughout the program.
- It describes in greater detail the specific features of a part of the program.

```
# Documentation: Saving documents
# 'save': save document under the current name
# 'save as' rename the document to a new name
<program statement>
<program statement>

# Documentation: Spell checking
# The program can spell check documents using the following English variants:
# English (British), English (American), English (Canadian)
<program statement>
<program statement>
```

Input

- The computer program getting *numeric information* from the user

- **Format:**

  <*variable name*> = input()

  OR

  <*variable name*> = input("<*Prompting message*>")

- **Example (the online program is called 'input1.py"):**

```
print "Type in a number: ",
num = input ()
          OR
num = input ("Type in a number: ")
```

The comma puts the string and the prompt on the same line.

Raw_Input

•Used to get *string input* (series of alphanumeric and other types of characters) from the user.

•Strings cannot be used for calculations.

•**Format:**

```
<variable name> = raw_input()
        OR
<variable name> = input("<Prompting message>")
```

•**Example (the online program is called 'input2.py"):**

```
 print "Enter your name: ",
 name = raw_input ()
OR
 name = raw_input ("Enter your name: ")
```

James Tam

Converting Between Different Types Of Information

•Example motivation: you may want numerical information to be stored as a string (for the formatting capabilities) but also you want that same information in numerical form (in order to perform calculations).

•Two conversion mechanisms available in Python:

Format:

```
int (<value to convert>)
float (<value to convert>)
```

Examples:

(Truncation: online example is called "convert1.py")

```
x = 10.9
y = int (x)
print x, y
```

James Tam

Converting Between Different Types Of Information (2)

Examples:

(String to numeric: online example is called "convert2.py")

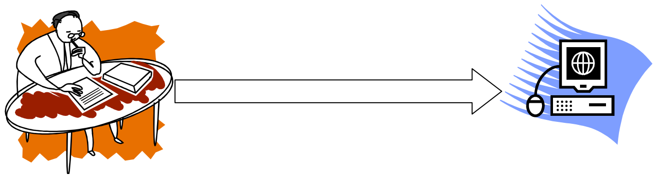
```
x = '100'
y = '-10.5'
print x + y
print int(x) + float (y)
```

Prewritten Python Functions

- Python comes with many functions that are a built in part of the language.
- (If a program needs to perform a quick task e.g., finding the absolute value of a number, then you should first check if the function has already been implemented).
- For a list of all prewritten Python functions.
  - http://docs.python.org/library/functions.html

The Translation Process: Reexamined

•What you should know: computer programs must be translated into machine language prior to execution.



•What you will learn: the two different ways/times that translation occurs.

Translation: Interpreting Vs. Compiling

1) Interpreters

- Each time that the program is run the interpreter translates the program (translating a part at a time).
- If there are any errors (syntax) during the process of interpreting the program, the program will stop running right when the error is encountered.

2) Compilers

- Before the program is run the compiler translates the program (compiling it all at once).
- If there are *any errors* (syntax) during the compilation process, no machine language executable will be produced.
- If there are *no errors* (syntax) during compilation then the translated machine language program can be run.

Types Of Programming Errors

1. Syntax/translation errors
2. Runtime errors
3. Logic errors

1. Syntax/ Translation Errors

- Each language has rules about how statements are to be structured.
- An English sentence is structured by the grammar of the English language:
  - The cat sleeps the sofa.

Grammatically incorrect: missing the preposition to introduce the prepositional phrase 'the sofa'

- Python statements are structured by the syntax of Python:
  - $\underbrace{5}$ = num

Syntactically incorrect: the left hand side of an assignment statement cannot be a literal (unnamed) constant.

1. Syntax/ Translation Errors (2)

- The translator checks for these errors when a computer program is translated to machine language:
  - For compiled programs (e.g., C, C++, Pascal) translation occurs once before the program is executed (because compilation occurs all at once before execution).
  - For interpreted programs (e.g., Python) translation occurs as each statement in the program is executing (because interpreting occurs just before each statement executes).

1. Some Common Syntax Errors

- Miss-spelling names of keywords
  - e.g., 'primt' instead of 'print'
- Forgetting to match closing quotes or brackets to opening quotes or brackets.
- Using variables before they've been named (allocated in memory). The online version of this program is called "syntax.py"

```
print num
```

2. Runtime Errors

- Occur as a program is executing (running).
- The syntax of the language has not been violated (each statement follows the rules/syntax).
- During execution a serious error is encountered that causes the execution (running) of the program to cease.
- With a language like Python where translation occurs just before execution (interpreted) the timing of when runtime errors appear won't seem different from a syntax error.
- But for languages where translation occurs well before execution (compiled) the difference will be quite noticeable.
- A common example of a runtime error is a division by zero error.

2. Runtime Error: An Example

- The online version of this program is called "runtime.py"

```
num2 = input("Type in a number: ")
num3 = input("Type in a number: ")
num1 = num2 / num3
print num1
```

3. Logic Errors

- The program has no syntax errors.
- The program runs from beginning to end with no runtime errors.
- But the logic of the program is incorrect (it doesn’t do what it’s supposed to and may produce an incorrect result).
- The online version of this program is called “logic.py”

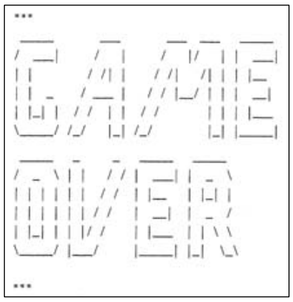
```
print "This program will calculate the area of a rectangle"
length = input("Enter the length: ")
width = input("Enter the width: ")
area = length + width
print "Area: ", area
```

Advanced Text Formatting

- Triple quoted output
- Using escape sequences

Triple Quoted Output

- •Used to format text output
- •The way in which the text is typed into the program is exactly the way in which the text will appear onscreen.
- •The online example program is called "formatting1.py"



From Python Programming (2nd Edition) by Michael Dawson

Escape Codes

- •The back-slash character enclosed within quotes won't be displayed but instead indicates that a formatting (escape) code will follow the slash:

Escape sequence	Description
\a	Alarm. Causes the program to beep.
\b	Backspace. Moves the cursor back one space.
\n	Newline. Moves the cursor to beginning of the next line.
\t	Tab. Moves the cursor forward one tab stop.
\'	Single quote. Prints a single quote.
\"	Double quote. Prints a double quote.
\\	Backslash. Prints one backslash.

Escape Codes (2)

• The online program is called “formatting2.py”

```
print "\a*Beep!*"
print "h\bello"
print "hi\nthere"
print 'it\'s'
print "he\\y \"you\" "
```

After This Section You Should Now Know

• How to create, translate and run Python programs.

• Variables:
  - What they are used for
  - How to access and change the value of a variable
  - Conventions for naming variables

• Named constants:
  - What are named constants and how they differ from variables
  - What are the benefits of using a named constant vs. a literal

• What is program documentation and what are some common things that are included in program documentation

• How are common mathematical operations performed

• Output:
  - How to display messages that are a constant string or the value of a memory location (variable or constant) onscreen with print

After This Section You Should Now Know (2)

- Input:
  - How to get a program to acquire and store information from the user of the program
  - How to get numeric vs. string input
- How do the precedence rules/order of operation work in Python
- How to convert between different types of information
- About the existence of prewritten Python functions and how to find descriptions of them
- What are the three programming errors, when do they occur and what is the difference between each one
- How triple quotes can be used in the formatting of output
- What is an escape code and how they can affect the output or execution of a program
- The difference between interpreted and compiled programs