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, Electronic Arts, some UNIX scripts etc.)
- Named after a British comedy "Monty Python's Flying Circus"
- Official website (Python the programming language, not the Monty Python comedy troop): http://www.python.org

Slide # 3

Online Help

- SAFEST APPROACH for working at home (**recommended**).
  - Remotely login to the Computer Science network
  - Example: Connect using a remote login program such as SSH
    - Info: http://pages.cpsc.ucalgary.ca/~tamj/231/starting/ssh.html)
    - Download: http://www.ucalgary.ca/it/software/downloads (SSH comes with MacOS so no download is needed. Alternative programs such as Fugu or Apple Terminal may be downloaded and installed).
  - (SSH needs to be installed but it is far easier to install SSH than it is to install Python).
- Alternative (**not recommended**): Getting Python (*get version 3.X* and not version 2.X)
  - http://www.python.org/download/
  - Requires that Python is configured (the "path") on your computer (it is *not mandatory to install Python at home*, follow these instructions carefully, missteps occur at your own peril!)
    - http://docs.python.org/using/windows.html
    - http://docs.python.org/using/unix.html
    - http://docs.python.org/using/mac.html

James Tam

Slide # 4

Online Help (2)

- (If you have installed Python on your own computer and still can't get 'Python' to run – this approach works although it's a 'inelegant' solution).
  - Note where you installed Python (folder or directory)
  - Create and run your Python programs from this location.

James Tam

Slide # 5

Online Help (3)

- Explanation of concepts (for beginners: along with examples to illustrate)
  - http://docs.python.org/py3k/tutorial/index.html
- Information about Python libraries (more advanced: useful for looking up specific details of pre-created Python functions after you have a general idea of how things work e.g., what is the exact wording of the function used to open a file).
  - http://docs.python.org/py3k/library/index.html
- General help (includes the above two help links and more):
  - http://docs.python.org/py3k/

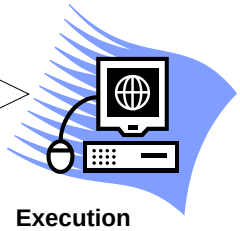
James Tam

Slide # 6

The Process Of Creating A Computer Program

Translation

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



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.

James Tam

Slide # 7

Location Of My Online Examples

- Finding them via the WWW:
  - URL: http://pages.cpsc.ucalgary.ca/~tamj/231/examples/
- Finding them in UNIX when you are logged onto a computer in the lab (or remotely logged in using a program like SSH)
  - Directory: /home/231/examples
- The locations of the example programs that are specific for this section of notes (each section will have be located in a sub-directory/sub-link):
  - http://pages.cpsc.ucalgary.ca/~tamj/231/examples/intro
  - /home/231/examples/intro

James Tam

Slide # 8

An Example Python Program

- Program name: small.py

Filename: small.py

```
print ("hello")
```

James Tam

Slide # 9

Creating Programs: One Operating System

- To translate/run your program type "Python <filename.py>" at the command line.
  - The first example program would be executed by typing "python small.py"
  - For a program whose filename is called "output1.py" you would type "python output1.py".

```
O:\231\examples\intro>python small.py
hello

O:\231\examples\intro>
```

  - If you didn't catch it: Make sure the exact and complete filename is entered (even the "dot-py" suffix).
- The computers in the Computer Science lab have already been set up so don't need to do any installing or preconfiguring of Python.
  - This applies even if you work remotely from home using a program such as SSH.
  - **On the Computer Science computers**: make sure you are running the correct version of python by typing "python3" (not just "python")
    - E.g., "python3 small.py"

James Tam

Slide # 10

Creating Programs: One Operating System (2)

- **Working on your own computer**: When you translate/run your program in the command window make sure that your command line is in the same location as your Python program ('inelegant but works').

```
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 location.

- Alternatively you have set up your computer so it 'knows' where python has been installed (e.g., setting the 'path' in Windows).
  - See earlier slide: "…follow these instructions carefully, missteps occur at your own peril!"
  - The computers in the CPSC lab have already been set up properly.

James Tam

Slide # 11

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.
  - Don't mix and match different types of quotation marks.
- **Format:**

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

  OR

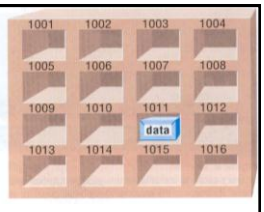
  print ('the message that you wish to appear')
- **Example:**

  print ("foo")

  print ('bar')

James Tam

Variables



- Set aside a location in memory.
- Used to store information (temporary).
  - This location can store one 'piece' of information.
    - Putting another piece of information at an existing location overwrites previous 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 include: integer (whole), floating point (fractional), strings (text)

  Format:

  <name of variable> = <Information to be stored in the variable>

  Examples:

  - Integer (e.g., num1 = 10)
  - Floating point (e.g., num2 = 10.0)
  - Strings (e.g., name = "james")

James Tam

Picture from Computers in your future by Pfaffenberger B

Slide # 13

Variable Naming Conventions

- Python requirements:
  - Rules built into the Python language for writing a program.
  - Somewhat analogous to the grammar of a 'human' language.
  - If the rules are violated then the typical outcome is the program cannot be translated (nor run).
    - A language such as Python may allow for a partial execution.
- Style requirements:
  - Approaches for producing a well written program.
  - (The real life analogy is that something written in a human language may follow the grammar but still be poorly written).
  - If they are not followed then the program can still be translated but there may be other problems (more on this during the term).

James Tam

Slide # 14

Variable Naming Conventions (2)

1. Style requirement: The name should be meaningful.
2. Style and Python requirement: Names *must* start with a letter (Python requirement) and *should not* begin with an underscore (style requirement).
3. Style requirement: Names are case sensitive but avoid distinguishing variable names only by case.
4. Style requirement: Variable names should generally be all lower case (see next point for the exception).
5. 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 don't mix and match.)
6. Python requirement: Can't be a keyword (see next slide).

James Tam

Slide # 15

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

James Tam

Slide # 16

Extra Practice

- Come up example names that violate and conform to the naming conventions.
  - (You will have to go through this process as you write your programs so it's a good idea to take about 5 – 10 minutes to make sure that you understand the requirements).

James Tam

Slide # 17

Named Constants

- They are similar to variables: a memory location that's been given a name.
- Unlike variables their contents *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).
- Example PI = 3.14
- 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.

James Tam

Slide # 18

Terminology: Named Constants Vs. Literals

- Named constant: given an explicit name.

```
TAX_RATE = 0.2
afterTax = income – (income * TAX_RATE)
```

- Literal/unnamed constant/magic number: not given a name, the value that you see is literally the value that you have.

```
afterTax = 100000 – (100000 * 0.2)
```

James Tam

Slide # 19

Terminology: Named Constants Vs. Literals

- Named constant: given an explicit name

```
TAX_RATE = 0.2
afterTax = income – (income * TAX_RATE)
```

Named constants

- Literal/unnamed constant/magic number: not given a name, the value that you see is literally the value that you have.

```
afterTax = 100000 – (100000 * 0.2)
```

Literals

James Tam

Slide # 20

Why Use Named Constants

1. They make your program easier to read and understand

```
# NO
populationChange = (0.1758 – 0.1257) * currentPopulation
```

Vs.

```
#YES
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!)[1]

Magic numbers: 'pulled out of no-where'

3.14 2.54 300,000

James Tam

Slide # 21

Why Use 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.
- Using named constants is regarded as "good" style when writing a computer program.

James Tam

Slide # 22

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
```

James Tam

Slide # 23

Purpose Of Named Constants (3)

```
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
```

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

James Tam

Slide # 24

Purpose Of Named Constants (4)

```
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
```

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

James Tam

Slide # 25

When To Use A Named Constant?

- (Rule of thumb): If you can assign a descriptive useful, self-explanatory name to a constant then you probably should.
- Example 1 (easy to provide self explanatory name)

```
INCH_CENTIMETER_RATIO = 2.54
height = height * INCH_CENTIMETER_RATIO
```

- Example 2 (providing self explanatory name is difficult)

```
Calories used = (10 x weight) + (6.25 x height) - [(5 x age) - 161]
```

James Tam

Slide # 26

Extra Practice

- Provide a formula where it would be appropriate to use named constants (should be easy).
- Provide a formula where unnamed constants may be acceptable (may be trickier).
- Search online if you can't think of any.

James Tam

Slide # 27

Output: Displaying The Contents Of Variables And Constants

Format:

print (<*variable name*>)
print (<*constant name*>)

Example:

Program name: output1.py

```
aNum = 10
A_CONSTANT = 10
print (aNum)
print (A_CONSTANT)
```

James Tam

Slide # 28

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:**

  Program name: 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.

James Tam

Slide # 29

Output: Problems

- Sometimes Python automatically adds an additional newline
- Name of example: output3.py

```
year = 1997
print ("year=")
print (year)
```

Label and variable contents on different lines

```
print ("year=", year)
```

Label and variable contents on the same line

James Tam

Slide # 30

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

James Tam

Slide # 31

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

James Tam

Slide # 32

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).

James Tam

Slide # 33

Input

- The computer program getting *string information* from the user.
- Strings cannot be used for calculations (information getting numeric input will provided shortly).

- **Format:**

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

- **Example:**

Program name: input1.py

```
print ("What is your name: ")
name = input ()
        OR
name = input ("What is your name: ")
```

James Tam

Slide # 34

Variables: Storing Information

- On the computer all information is stored in binary (2 states)
  - Example: RAM/memory stores information in a series of on-off combinations

Bit  on OR off

Byte 

- 8 bits

James Tam

Slide # 35

Variables: Storing Information (2)

- Information must be converted into binary to be stored on a computer.

User enters → Can be stored in the computer as

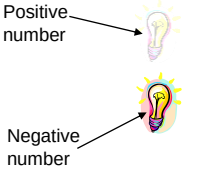
13

James Tam

Slide # 36

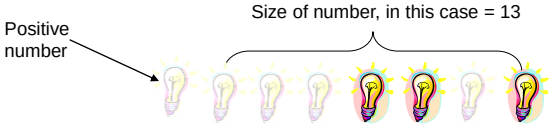
Storing Integer Information

- 1 bit is used to represent the sign, the rest is used to store the size of the number
  - Sign bit: 1/on = negative, 0/off = positive
- **Format:**



Digits representing the size of the number

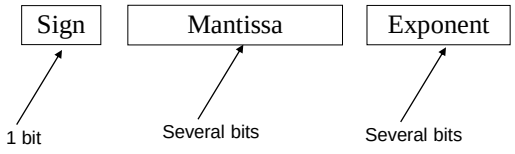
- **Previous example**



James Tam

Slide # 37

Storing Real Numbers In The Form Of Floating Point



- Mantissa: digits of the number being stored
- Exponent: the direction and the number of places the decimal point must move ('float') when storing the real number as a floating point value.

- Examples with 5 digits used to represent the mantissa:
  - e.g. One: 123.45 is represented as $12345 * 10^{-2}$
  - e.g. Two: 0.12 is represented as $12000 * 10^{-5}$
  - e.g. Three: 123456 is represented as $12345 * 10^{1}$
- Remember: Using floating point numbers may result in a loss of accuracy (the float is an approximation of the real value to be stored).

James Tam

Slide # 38

Storing Character Information

- Typically characters are encoded using ASCII
- Each character is mapped to a numeric value
  - E.g., 'A' = 65, 'B' = 66, 'a' = 97, '2' = 50
- These numeric values are stored in the computer using binary

Character	ASCII numeric code	Binary code
'A'	65	01000001
'B'	66	01000010
'a'	97	01100001
'2'	50	00110010

James Tam

Slide # 39

Storing Information: Bottom Line

- Why it important to know that different types of information is stored differently?
- Certain operations only apply to certain types of information and can produce errors or unexpected results when applied to other types of information.
- **Example**

```
num = input("Enter a number")
numHalved = num / 2
```

James Tam

Slide # 40

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).
- Some of the conversion mechanisms available in Python:

Format:

int (<*value to convert*>)
float (<*value to convert*>)
str (<*value to convert*>)

Examples:

Program name: convert1.py

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

James Tam

Slide # 41

Converting Between Different Types Of Information (2)

Examples:

Program name: convert2.py

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

(Numeric to string: convert3.py)

```
aNum = 123
aString = str(aNum)
aNum = aNum + aNum
aString = aString + aString
print (aNum)
print (aString)
```

James Tam

Slide # 42

Converting Between Different Types Of Information: Getting Numeric Input

- Because the 'input' function only returns string information it must be converted to the appropriate type as needed.
  - **Example**

    Program name: convert4.py

 # Problem!

```
HUMAN_CAT_AGE_RATIO = 7
age = input("What is your age in years: ")
catAge = age * HUMAN_CAT_AGE_RATIO
print ("Age in cat years: ", catAge)
```

- **'Age' refers to a string not a number.**
- **The '*' is not mathematical multiplication**

 # Problem solved!

```
HUMAN_CAT_AGE_RATIO = 7
age = int(input("What is your age in years: "))
catAge = age * HUMAN_CAT_AGE_RATIO
print ("Age in cat years: ", catAge)
```

- **'Age' converted to an integer.**
- **The '*' now multiplies a numeric value.**

James Tam

Slide # 43

Determining The Type Of Information Stored In A Variable

- It can be done by using the pre-created python function 'type'
- Example program: type.py

```
myInteger = 10
myString = "foo!"
print (type(myInteger))
print (type(10.5))
print (type(myString))
```


Slide # 44

Output: Formatting

- Output can be formatted in Python through the use of placeholders.
- **Format**:

  print ("%<*type of info to display/code*>" %<*source of the info to display*>)

- **Example**:
  - Program name: formatting1.py

```
num = 123
st = "cpsc 231"
print ("num=%d"       %num)
print ("course: %s"   %st)
num = 12.5
print ("%f %d" %(num, num))
```

Slide # 45

Types Of Information That Can Be Displayed

Descriptor code	Type of Information to display
%s	String
%d	Integer (d = decimal / base 10)
%f	Floating point

James Tam

Slide # 46

Some Formatting Effects Using Descriptor Codes

- **Format**:

  %<*width*>[1].<*precision*>[2]<*type of information*>

- **Examples**:
  - Program name: formatting2.py

```
num = 12.55
print ("%5.1f" %num)
print ("%.1f" %num)
num = 12
st = "num="
print ("%s%d" % (st, num))
print ("%5s%5s%1s" % ("hi", "hihi", "there"))
```

1 A positive integer will add leading spaces (right align), negatives will add trailing spaces (left align). Excluding a value will set the field width to a value large enough to display the output

2 For floating point representations only.

James Tam

Slide # 47

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.
- Program name: formatting3.py

From Python Programming (2nd Edition) by Michael Dawson

Slide # 48

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.
\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.

Slide # 49

Escape Codes (2)

- Program name: formatting4.py

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


Slide # 50

Extra Practice

- Traces:
  - Modify the examples (output using descriptor and escape codes) so that they are still valid Python statements.
    - Alternatively you can try finding some simple ones online.
  - Hand trace the code (execute on paper) without running the program.
  - Then run the program and compare the actual vs. expected result.
- Program writing:
  - Write a program the will right-align text into 3 columns of data.
  - When the program starts it will prompt the user for the maximum width of each column

Slide # 51

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.

James Tam

Slide # 52

Program Documentation (2)

- It doesn't contain instructions for the computer to execute.
- It doesn't get translated into machine language.
- 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).

James Tam

Slide # 53

Program Documentation (3)

- **Format:**

  # <Documentation>

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

- **Examples:**

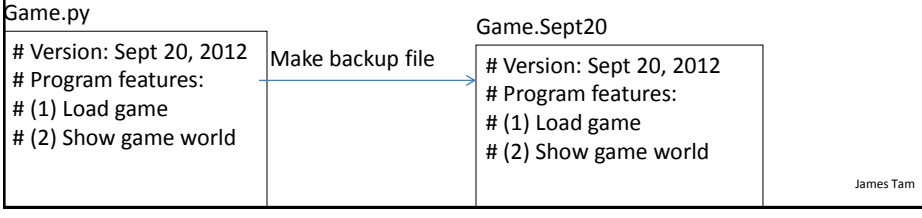
  # Tax-It v1.0: This program will electronically calculate your tax return.
  # This program will only allow you to complete a Canadian tax return.

James Tam

Slide # 54

Program Versioning And Back Ups

- As significant program features have been completed (tested and the errors removed/debugged) a new version should be saved in a separate file.

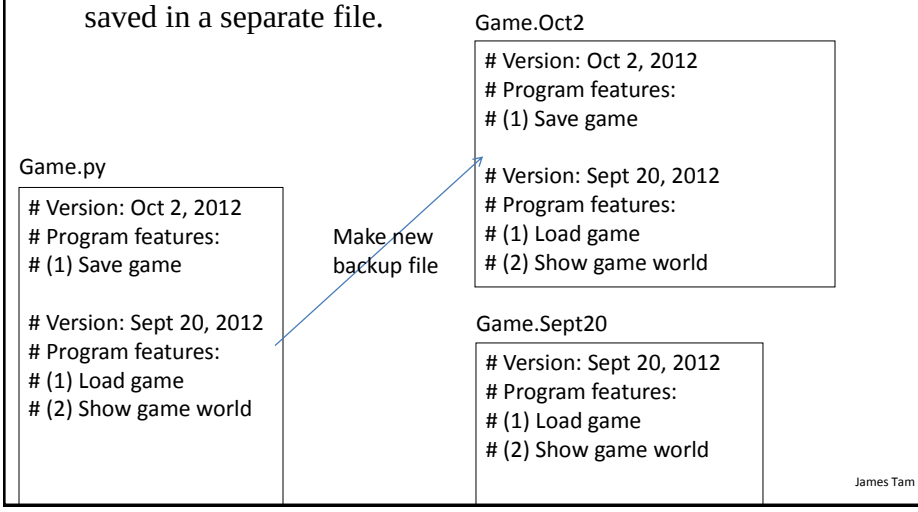


James Tam

Slide # 55

Program Versioning And Back Ups

- As significant program features have been completed (tested and the errors removed/debugged) a new version should be saved in a separate file.



James Tam

Slide # 56

Backing Up Your Work

- Do this every time that you have completed a significant milestone in your program.
  - What is 'significant' will vary between people but make sure you do this.
- Ideally the backup file should be stored in a separate directory/folder (better yet on a separate device and/or using an online method such as an email attachment).
- Common student reason for not making copies: "Backing up files takes time to do!"
- Compare:
  - Time to copy a file: ~10 seconds (generous in some cases).
  - Time to re-write your program to implement that feature again: 10 minutes (might be overly conservative in some cases).
- **Failing to backup your work is not a sufficient reason for receiving an extension**.

James Tam

Slide # 57

Types Of Documentation

- Header documentation
- Inline documentation

James Tam

Slide # 58

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>
```

James Tam

Slide # 59

Inline Documentation

- Provided throughout the program.
- It describes in greater detail the specific features of a part of the program (function, loop, branch, group of related statements).

```
# 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>
```

James Tam

Slide # 60

Prewritten Python Functions

- Python comes with many functions that are a built in part of the language e.g., 'print', 'input'
- (If a program needs to perform a common 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
- Note: some assignments may have specific instructions on which functions you are allowed to use.
  - Read the requirements specific to each assignment.

James Tam

Slide # 61

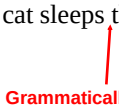
Types Of Programming Errors

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

Slide # 62

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:

```
5 = num
```

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

Slide # 63

1. Syntax/ Translation Errors (2)

- The translator checks for these errors when a computer program is translated to machine language.

James Tam

Slide # 64

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).
- Program name: error_syntax.py

```
print (num)
num = 123
print num
```

James Tam

Slide # 65

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.

James Tam

Slide # 66

2. Runtime Error[1]: An Example

- Program name: error_runtime.py

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

1 When 'num3' contains zero

James Tam

Slide # 67

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).
- Program name: error_logic.py

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

James Tam

Slide # 68

Some Additional Examples Of Errors

- All external links (not produced by your instructor):
  - http://level1wiki.wikidot.com/syntax-error
  - http://www.cs.bu.edu/courses/cs108/guides/debug.html
  - http://cscircles.cemc.uwaterloo.ca/1e-errors/
  - http://www.greenteapress.com/thinkpython/thinkCSpy/html/app01.html

James Tam

Slide # 69

Practice Exercise

- •(This one will be an ongoing task).
- •As you write you programs, classify the type of errors that you
- face as: syntax/translation, runtime or logical.

James Tam

Slide # 70

Layout And Formatting

- Similar to written text: all computers programs (except for the smallest ones) should use white space to group related instructions and to separate different groups.

```
# These are output statements to prompt the user information
Instruction1
Instruction2
Instruction3
Instruction4


# These are instructions to perform calculations on the user input and
display the results
Instruction5
Instruction6
```

James Tam

Slide # 71

The Squint Test: A Tool For Evaluating Layout

- Squint at the document or screen so that details (such as text) appear blurred.

Original webpage

Blurred version

- It's used to determine what stands out or what elements appear to belong together
  - The goal is to determine the overall structure by hiding details

Slide # 72

A Webpage That Fails The Squint Test

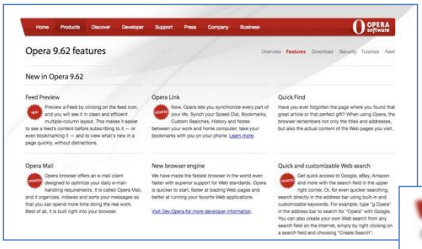
Original webpage

Blurred version

Images from: http://www.usabilitypost.com/

Slide # 73

A Webpage With Better Squint Test Results



Original webpage

Blurred version

Images from: http://www.usabilitypost.com/

James Tam

Slide # 74

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
  - How information is stored differently with different types of variables, converting between types
- Named constants:
  - What are named constants and how they differ from regular 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

James Tam

Slide # 75

After This Section You Should Now Know (2)

- Output:
  - How to display messages that are a constant string or the value stored in a memory location (variable or constant) onscreen with print
- How to format output through:
  - The use of descriptor codes.
  - Escape codes
- How triple quotes can be used in the formatting of output
- Input:
  - How to get a program to acquire and store information from the user of the program
- How do the precedence rules/order of operation work in Python
- About the existence of prewritten Python functions and how to find descriptions of them

James Tam

Slide # 76

After This Section You Should Now Know (3)

- What are the three programming errors, when do they occur and what is the difference between each one

James Tam