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"
  - http://www.python.org/doc/essays/ppt/fannie/ (Includes slides that provide an overview of Python as well as it's origins).
- Official website (Python the programming language, not the Monty Python comedy troop): http://www.python.org

Python History

- Developed in the early 1990's by Guido an Rossum.
- Python was designed with a tradeoff in mind (from "*Python for everyone*" (Horstman and Necaise):
  - Pro: Python programmers could quickly write programs (and not be burdened with an overly difficult language)
  - Con: Python programs weren't optimized to run as efficiently as programs written in some other languages.

"Gawky and proud of it."

From: http://www.python.org/~guido/

James Tam

Working At Home

- 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/217/starting/ssh_index.html)
    - Download: http://www.ucalgary.ca/it/software/downloads (SSH comes with MacOS so no download is needed.
    - (SSH still needs to be installed but it is far easier to install SSH than it is to install and setup 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

Working At Home (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.

Online Help: Official Python Site

- *Basic explanation* of concepts (for beginners: along with examples to illustrate)
  – http://docs.python.org/py3k/tutorial/index.html
- *More advanced 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/

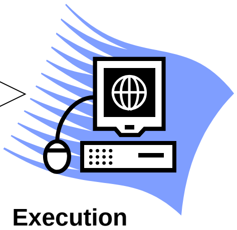
The Process Of Creating A Computer Program

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 (**e.g., Python**) are high level (look much like a human language).

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**)



Execution

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

Types Of Translators

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 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* during the compilation process, no machine language executable will be produced.
   - If there are *no errors* during compilation then the translated machine language program can be run.

Location Of My Online Examples

- Finding them via the WWW:
  - URL: http://pages.cpsc.ucalgary.ca/~tamj/217/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/courses/217/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/217/examples/intro
  - /home/courses/217/examples/intro

An Example Python Program

- Program name: small.py

Filename: small.py

```
print ("hello",end="")
```

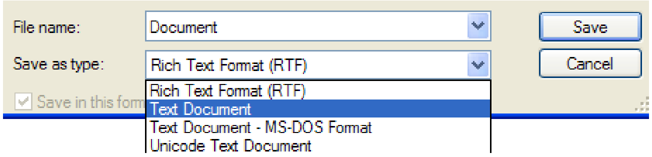
Creating/Running Programs: One Operating System (1)

- The process is similar on other platforms/OS's (the TA's will show you how to do it on the lab computers during tutorials).

Step 1: Writing your program

– 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** ending with the suffix dot-py "`.py`"



File name: Document

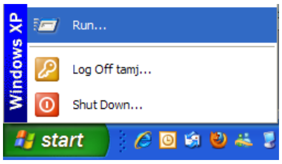
Save as type: Rich Text Format (RTF)

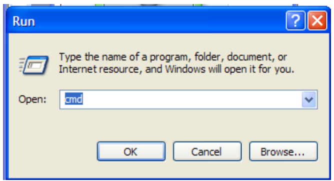
- Rich Text Format (RTF)
- Text Document
- Text Document - MS-DOS Format
- Unicode Text Document

Creating/Running Programs: One Operating System (2)

Step 2: Running your program

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





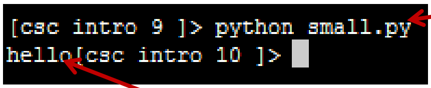
Run

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

Open: cmd

– 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`".



```
[csc intro 9 ]> python small.py
hello[csc intro 10 ]>
```

Running/translating program

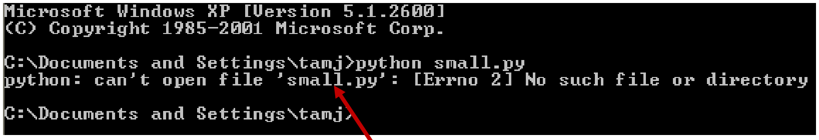
Output of program (result of running program)

Important Reminders

- Make sure you type the whole file name (including the part after the period) when you translate/run your program.
- In the computer lab (typing "python" runs Version 2)
  - "`python3 filename.py`"
- Working at home:
  - When you work remotely via a program such as SSH (recommended approach) type: "`python3 filename.py`"
  - Running Python on your home computer type (not recommended): "`python filename.py`"

Creating/Running Python Programs On Your Computer (1)

- **Running Python on your own computer (not mandatory for this course** : 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').
- Wrong approach:

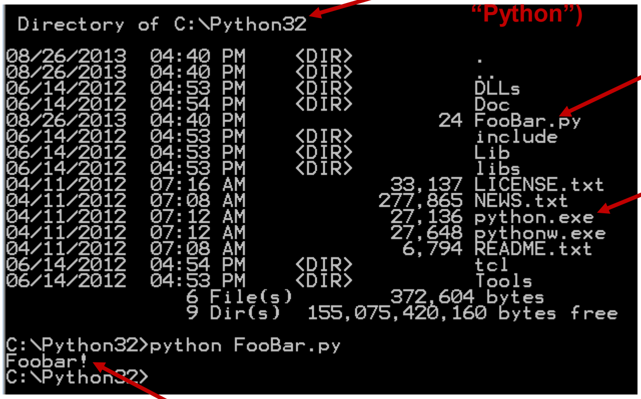


```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

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.

Creating/Running Programs Python On Your Computer (2)

- Correct approach:



```
 Directory of C:\Python32

08/26/2013  04:40 PM    <DIR>          .
08/26/2013  04:40 PM    <DIR>          ..
06/14/2012  04:53 PM    <DIR>          DLLs
06/14/2012  04:54 PM    <DIR>          Doc
08/26/2013  04:40 PM                24 FooBar.py
06/14/2012  04:53 PM    <DIR>          include
06/14/2012  04:53 PM    <DIR>          Lib
06/14/2012  04:53 PM    <DIR>          libs
04/11/2012  07:16 AM            33,137 LICENSE.txt
04/11/2012  07:08 AM           277,865 NEWS.txt
04/11/2012  07:12 AM            27,136 python.exe
04/11/2012  07:12 AM            27,648 pythonw.exe
04/11/2012  07:08 AM             6,794 README.txt
06/14/2012  04:54 PM    <DIR>          tcl
06/14/2012  04:53 PM    <DIR>          Tools
               6 File(s)        372,604 bytes
               9 Dir(s)  155,075,420,160 bytes free

C:\Python32>python FooBar.py
Foobar!
C:\Python32>
```

Location of the Python translation program “Python”)

Python program just written

Program to translate/run Python programs

Output of the Python program

James Tam

Creating/Running Programs Python On Your Computer (3)

- 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 so you don’t have do any configuring.

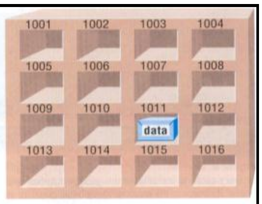
James Tam

Section Summary: Writing A Small "Hello World" Program

- You should know exactly what is required to create/run a simple yet functional Python program.
  - While you may not be able to create a new program at this point (but soon you will be able to do this) you should be able to enter/run a program like `small.py` yourself.

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 types of information which can be stored in variables include: integer (whole), floating point (fractional), strings (essentially any text you can type)

Format:

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

Examples:

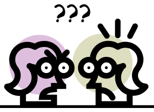
- Integer (e.g., `num1 = 10`)
- Floating point (e.g., `num2 = 10.0`)
- Strings: alpha, numeric, other characters enclosed in quotes.
  - e.g., `name = "james"`
  - Don't mix and match different types of quotation marks

Picture from Computers in your future by Pfaffenberger B

James Tam

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 (it runs until the error is encountered).
- 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 style requirements are not followed then the program can still be translated but there may be other problems (more on this during the term).



Variable Naming Conventions (2)

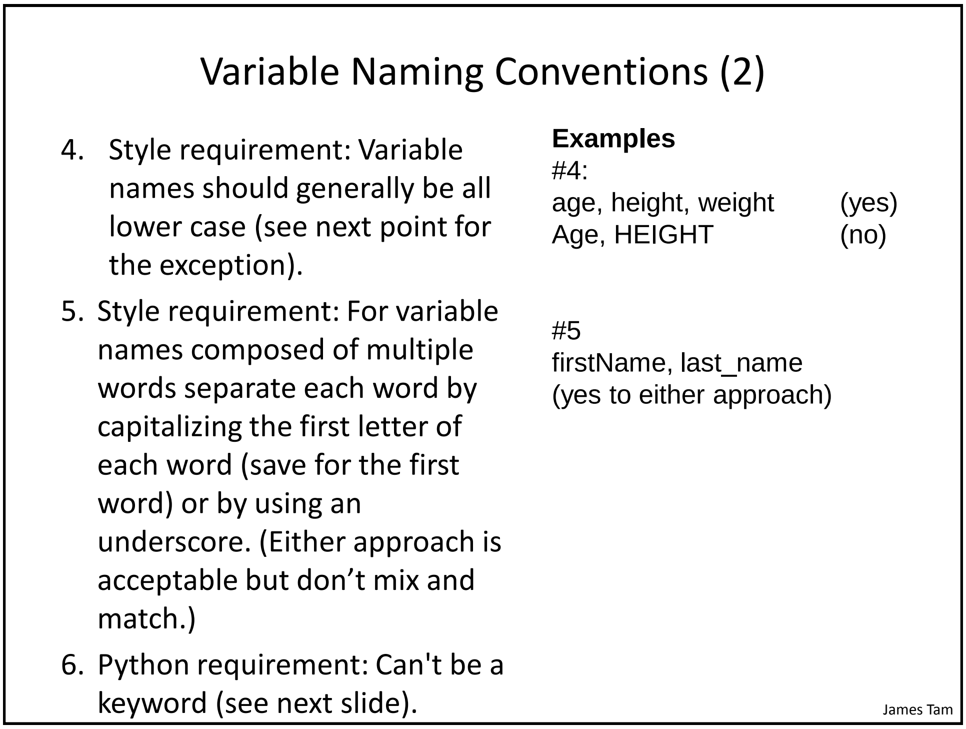
1. Style requirement: The name should be meaningful.
2. Style and Python requirement: Names *must* start with a letter (Python requirement) and *should not* begin with an underscore (style requirement).
3. Style requirement: Names are case sensitive but avoid distinguishing variable names only by case.

Examples

#1:
age (yes)   x, y(no)

#2
height (yes)   2x, _height (no)

#3
Name, name, nAme (no to this trio)



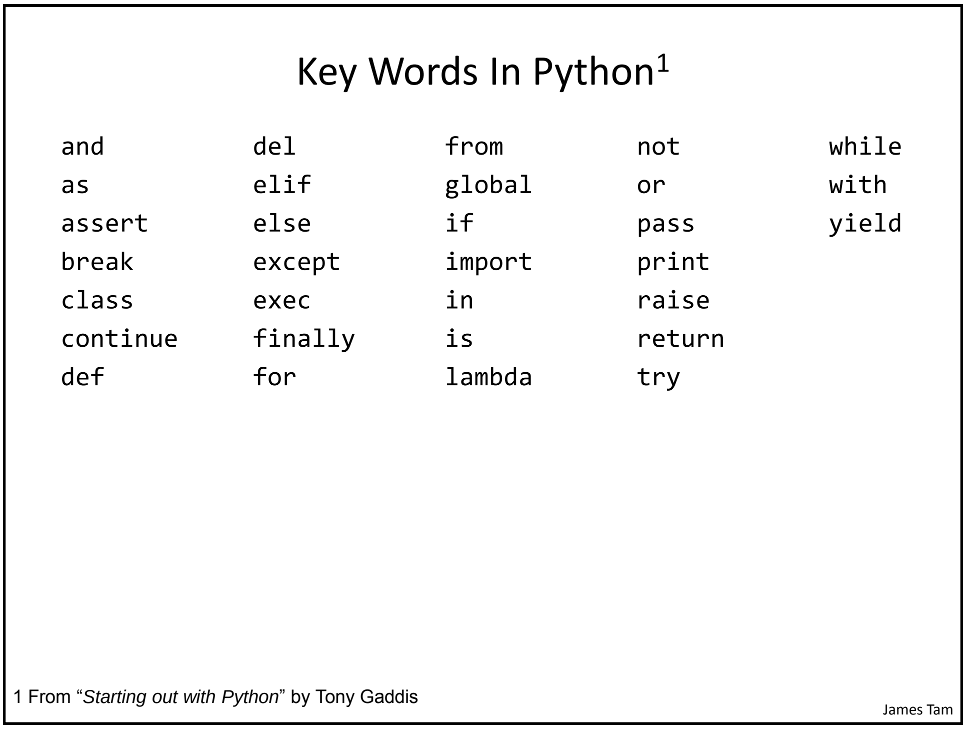
Variable Naming Conventions (2)

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).

Examples
#4:
age, height, weight (yes)
Age, HEIGHT (no)

#5
firstName, last_name
(yes to either approach)

James Tam



Key Words In Python[1]

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

[1] From “*Starting out with Python*” by Tony Gaddis

James Tam

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).

Section Summary: Variables

- What is a variable
- What types of variables exist in Python
- How to create a variable in Python
- What are naming conventions for variables

Displaying Output Using The Print() Function

- This function takes zero or more arguments (inputs)
  - Multiple arguments are separated with commas
  - `Print()` will display all the arguments followed by a blank line (move the cursor down a line).
    - end="" isn't mandatory but can be useful to prevent Python from adding the extra line (when precise formatting is needed)
  - Zero arguments just displays a blank line

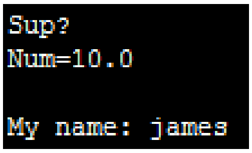
Format:

```
print (arg1, arg2 … )[1]
```

Example[2]: `output.py`

```
num = 10.0
name = 'james'
print("Sup?")
print("Num=", end="")
print(num)
print()
print('My name:', name)
```

Exercise: remove these and see if you can correctly predict the results.



```
Sup?
Num=10.0

My name: james
```

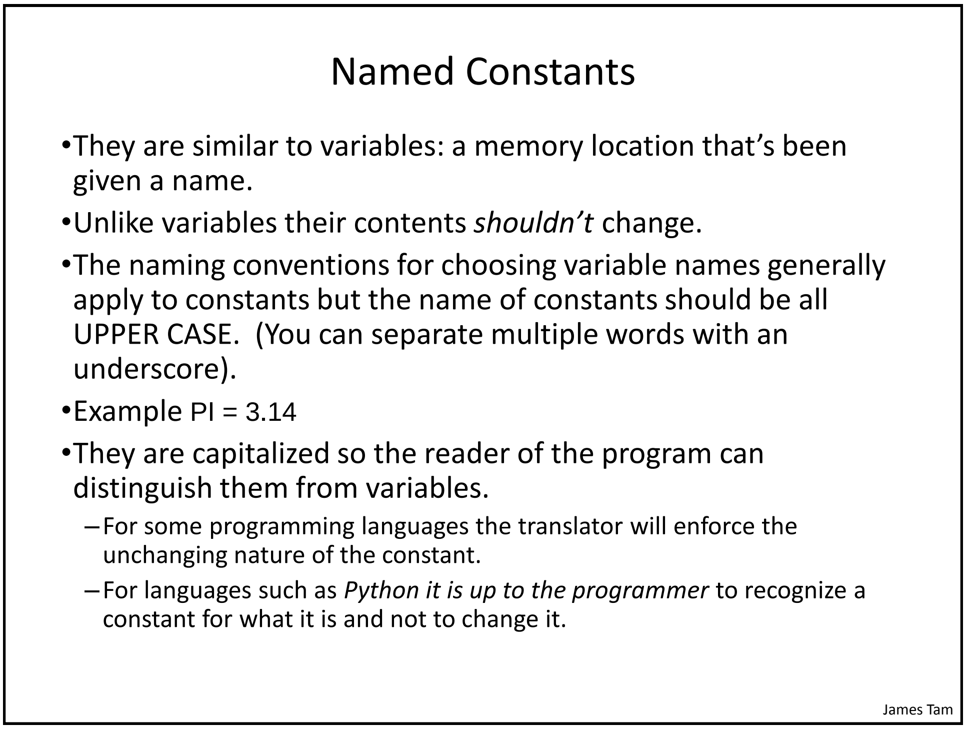
[1] From what you've learned thus far each argument can be a string or name of a variable.

Additional Note

- The assignment operator '=' used in programming languages does not have the same meaning as mathematics.
  - Don't mix them up!
- Example:

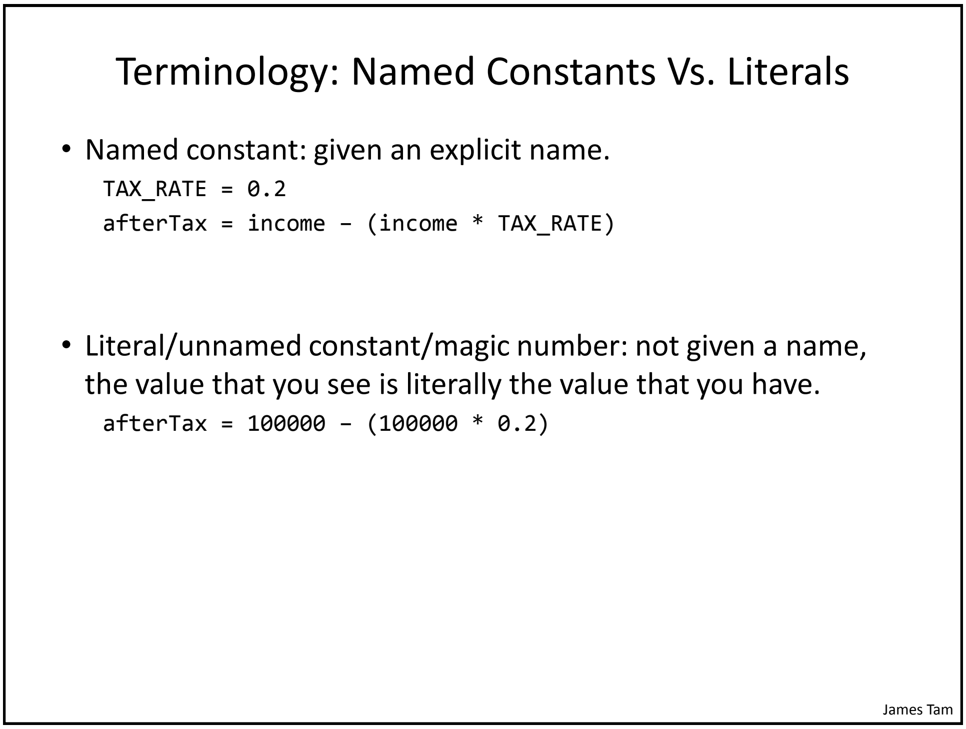
```
y = 3
x = y
x = 6
y = 13
```

- What is the end result? How was this derived (what are the intermediate results)?
- See program '`assignment.py`'



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.



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)
```

Terminology: Named Constants Vs. Literals

- Named constant: given an explicit name

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

Named constant

- 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

Why Use Named Constants

1. They make your program easier to read and understand

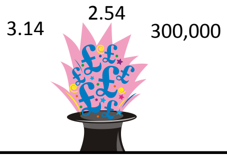
```
# NO
populationChange = (0.1758 – 0.1257) * currentPopulation
```

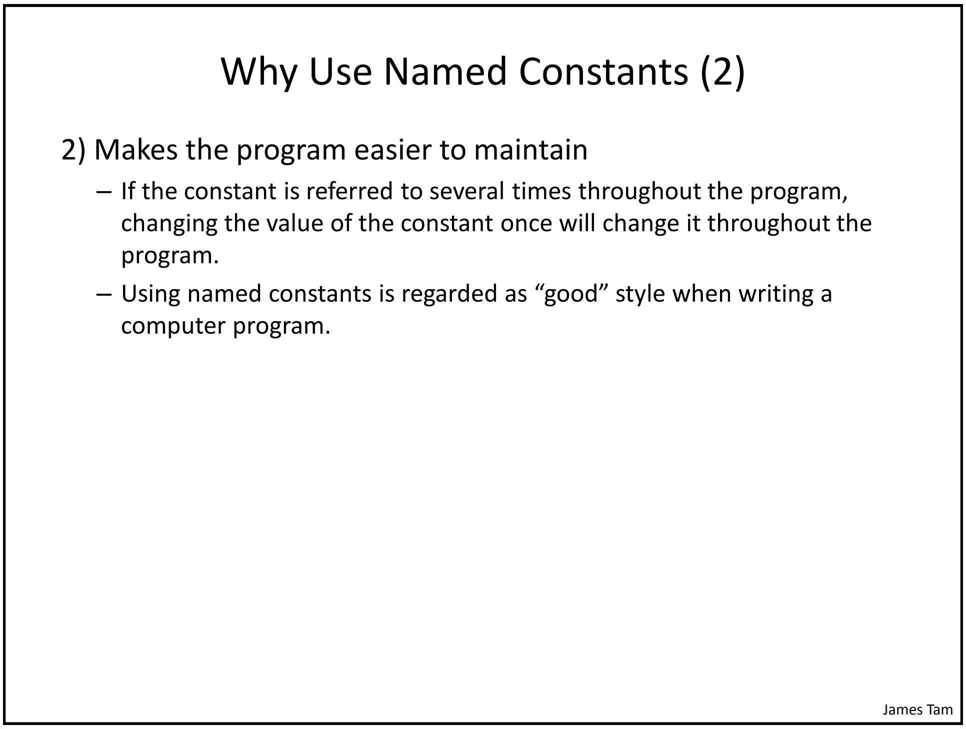
Vs.

In this case the literals are Magic Numbers (avoid whenever possible!)[1]

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

Another name for literals “Magic numbers”: ‘pulled out of no-where’



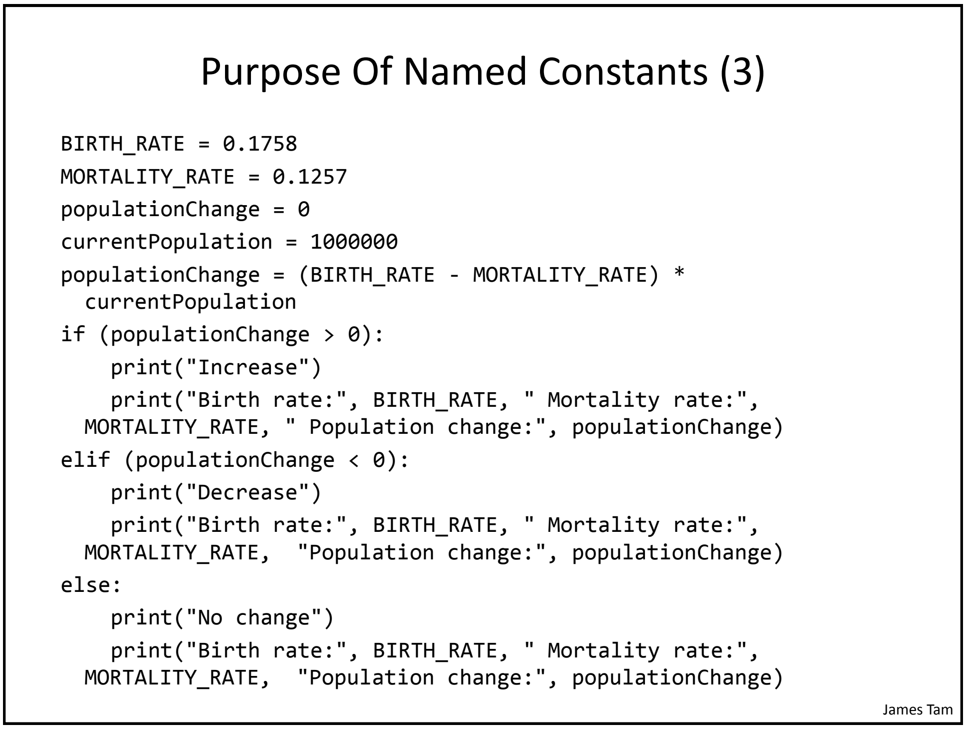


Why Use Named Constants (2)

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.

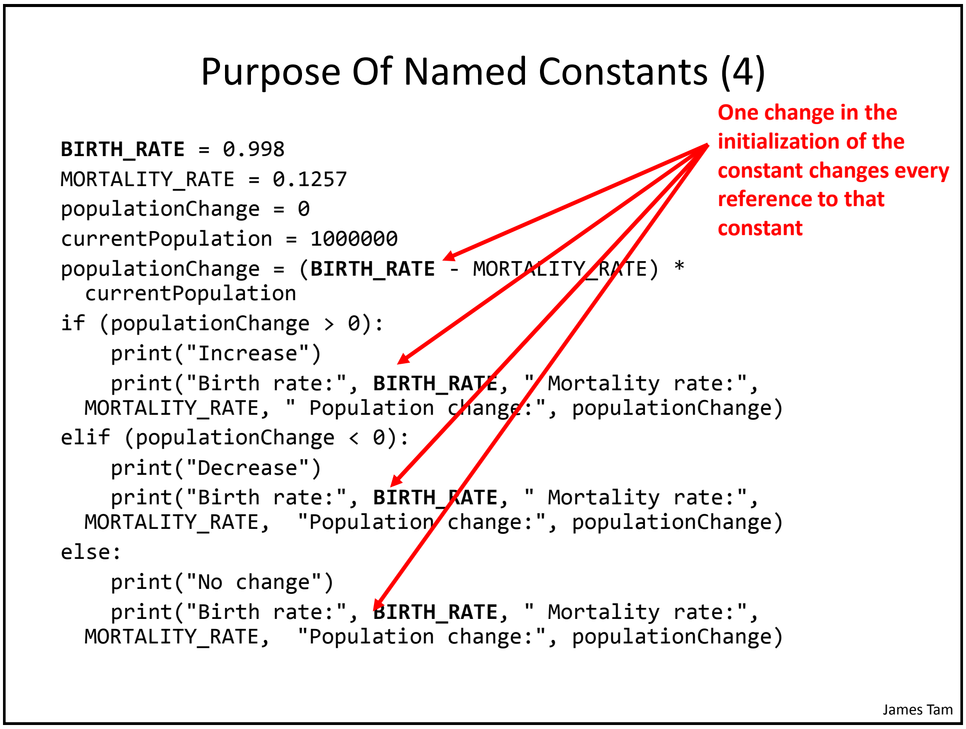
James Tam



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

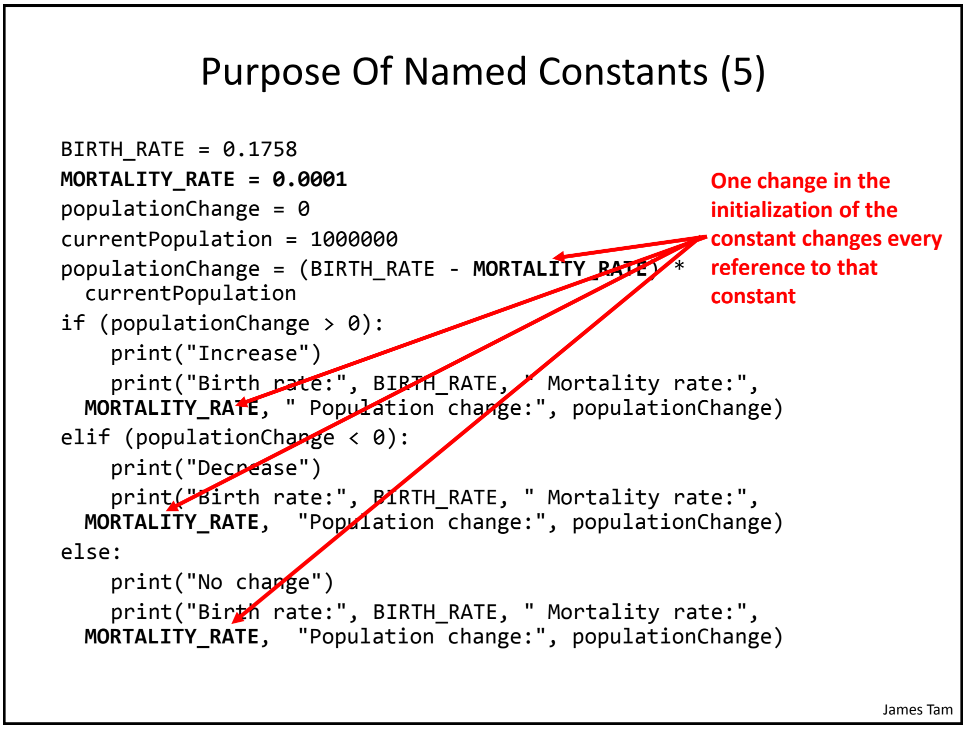


Purpose Of Named Constants (4)

```
BIRTH_RATE = 0.998
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



Purpose Of Named Constants (5)

```
BIRTH_RATE = 0.1758
MORTALITY_RATE = 0.0001
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

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 constant name)

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

- Example 2 (providing self explanatory constant name is difficult)

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

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.

Section Summary: Named Constants

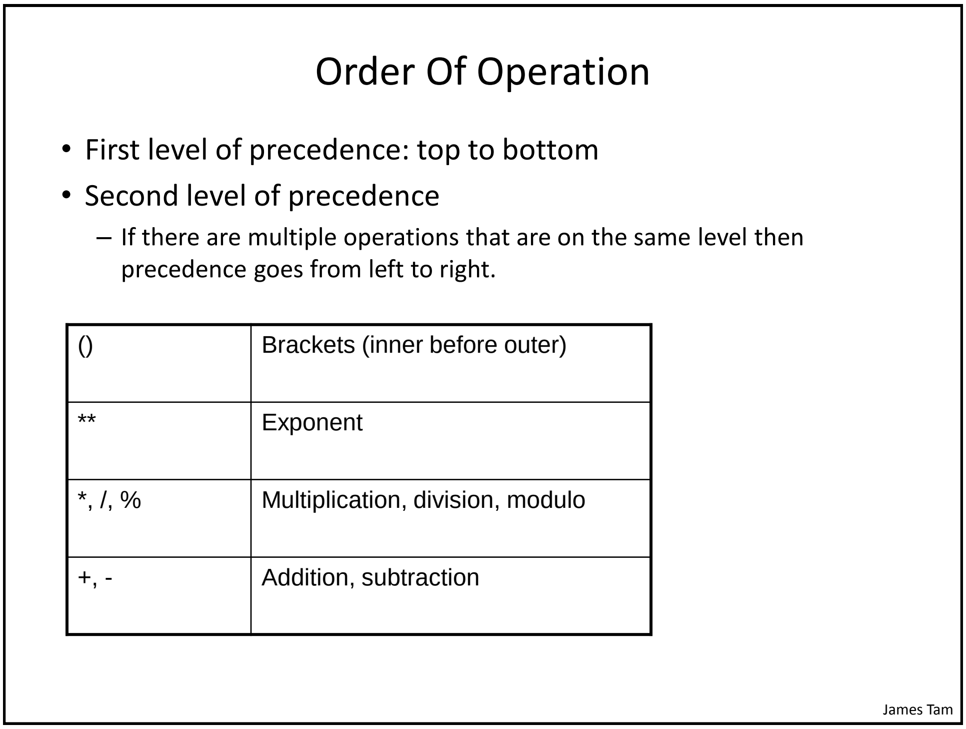
- What is a named constant
  - How does it differ from a variable
  - How does it differ from a literal/magic number
- What are some reasons for using named constants
- Naming conventions for named constants

James Tam

Arithmetic Operators

Operator	Description	Example	
=	Assignment	num **=** 7	
+	Addition	num = 2 **+** 2	
-	Subtraction	num = 6 **-** 4	
*	Multiplication	num = 5 ***** 4	
/	Division	num = 9 **/** 2	4.5
//	Integer division	num = 9 **//** 2	4
%	Modulo	num = 8 **%** 3	2
**	Exponent	num = 9 ****** 2	81

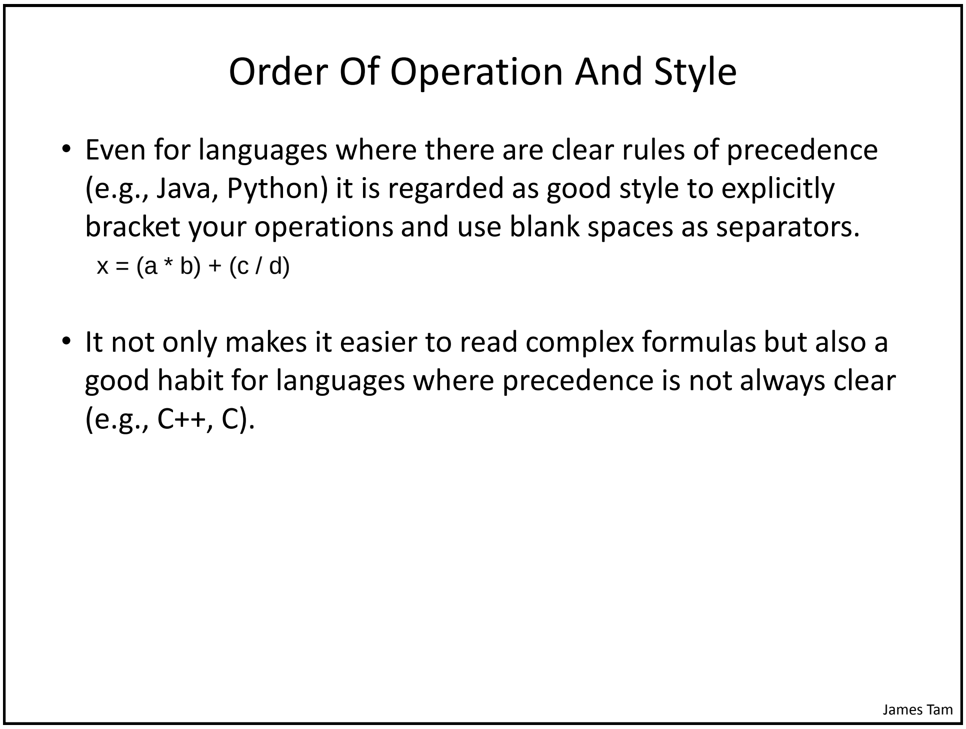
James Tam



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 and use blank spaces as separators.

```
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).

Input

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

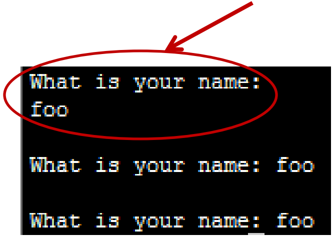
- **Format:**

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

Avoid alignment issues such as this



```
What is your name:
foo

What is your name: foo

What is your name: foo
```

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
  - A single off/off combination is referred to as a 'bit'

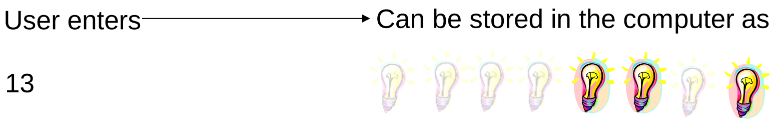
Bit on OR off

Byte

- 8 bits

Variables: Storing Information (2)

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



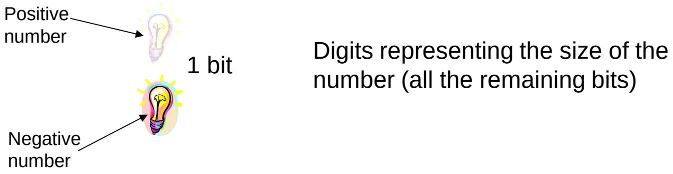
User enters → Can be stored in the computer as

13

James Tam

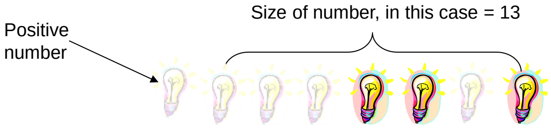
Storing Integer Information

- 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 (all the remaining bits)

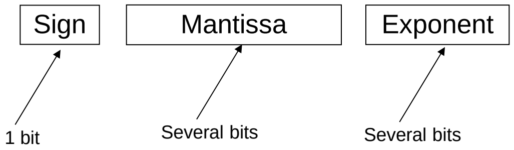
- **Previous example**



Size of number, in this case = 13

James Tam

Storing Real Numbers In The Form Of Floating Point



Sign

Mantissa

Exponent

1 bit

Several bits

Several bits

  - 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).

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

Storing Information: Bottom Line

- Why it important to know that different types of information is stored differently?
  - One motivation: sometimes students don’t why it’s significant that “123” is not the same as the number 123.
  - 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
```

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 (functions) 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)
```

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)
```

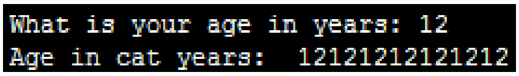
Converting Between Different Types Of Information: Getting Numeric Input (1)

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

    Program name: convert4.py

```
# No conversion performed: 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**

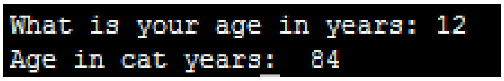


```
What is your age in years: 12
Age in cat years:  12121212121212
```

Converting Between Different Types Of Information: Getting Numeric Input (2)

```
# Input converted: 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.



```
What is your age in years: 12
Age in cat years:  84
```

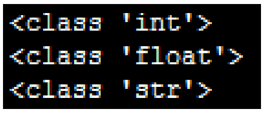
Section Summary: Input, Representations

- How to get user input in Python
- How do the different types of variables store/represent information
- How/why to convert between different types

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))
```

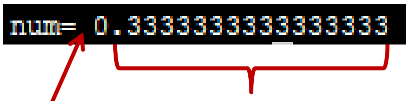


```
<class 'int'>
<class 'float'>
<class 'str'>
```

By Default Output Is Unformatted

- Example:

```
num = 1 / 3
print("num=",num)
```



```
num= 0.3333333333333333
```

Sometimes you get extra spaces (or blank lines)

The number of places of precision is determined by the language not the programmer

- There may be other issues e.g., you want to display output in columns of fixed width, or right/left aligned
- There may be times that it’s desirable to have a greater degree of control/precision over program output.

Formatting Output

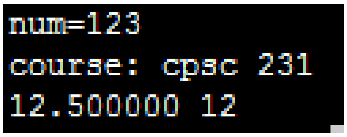
- Output can be formatted in Python through the use of placeholders.
- **Format**:

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

Don't literally display this: Placeholder (for information to be displayed)

- **Example (starting with simple cases)**:
  - 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))
```



```
num=123
course: cpsc 231
12.500000 12
```

Types Of Information That Can Be Formatted Via Descriptor Codes (Placeholder)

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

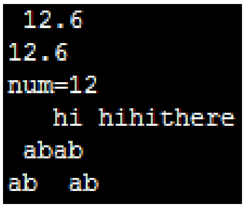
Some Formatting Effects Using Descriptor Codes

- **Format**:

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

- **Examples (more complex, illustrating application of codes)**:
  - Program name: `formatting2.p`

```
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"))
print ("%3s%-3s" %("ab", "ab"))
print ("%-3s%3s" %("ab", "ab"))
```



```
 12.6
12.6
num=12
   hi hihithere
 abab
ab  ab
```

[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.

Application Of Descriptor Codes

- It can be used to align columns of text.
- Example (movie credits)

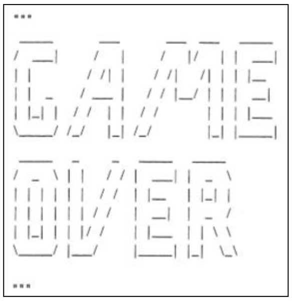
Section Summary: Formatting Output

- How to use descriptor codes (and field width, precision) to format output

James Tam

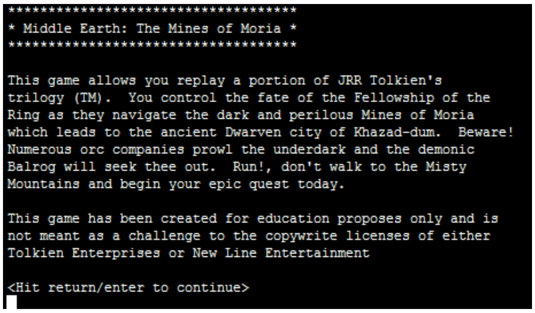
Triple Quoted Output

- Used to format text output (free form and to reduce the number of calls to the `print()` function)
- 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



```
************************************
* Middle Earth: The Mines of Moria *
************************************

This game allows you replay a portion of JRR Tolkien's
trilogy (TM).  You control the fate of the Fellowship of the
Ring as they navigate the dark and perilous Mines of Moria
which leads to the ancient Dwarven city of Khazad-dum.  Beware!
Numerous orc companies prowl the underdark and the demonic
Balrog will seek thee out.  Run!, don't walk to the Misty
Mountains and begin your epic quest today.

This game has been created for education proposes only and is
not meant as a challenge to the copywrite licenses of either
Tolkien Enterprises or New Line Entertainment

<Hit return/enter to continue>
```

James Tam

Escape Codes

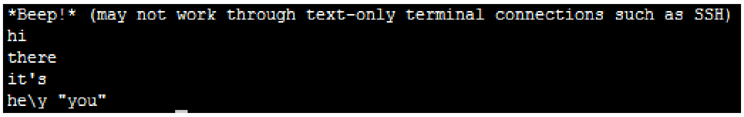
- The back-slash character enclosed within quotes won’t be displayed but instead indicates that a formatting (escape) code will follow the slash:

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.

Escape Codes (2)

- Program name: formatting4.py

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

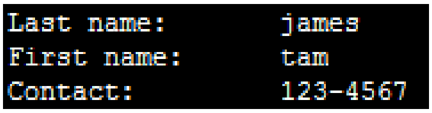


```
*Beep!* (may not work through text-only terminal connections such as SSH)
hi
there
it's
he\y "you"
```

Escape Codes: Application

- It can be used to nicely format text output.
- Program example: formatting5.py

```
firstName = input("First name: ")
lastName = input("Last name: ")
mobile = input("Cell phone number: ")
print("Last name:\t", lastName)
print("First name:\t", firstName)
print("Contact:\t", mobile)
```



```
Last name:        james
First name:       tam
Contact:          123-4567
```

Section Summary: Escape Codes

- How to use escape codes to format output

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 or a textbook.
  - 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

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 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 – more on this in a moment “Program versioning and backups”).

Program Documentation (3)

- **Format:**

```
# <Documentation>
```

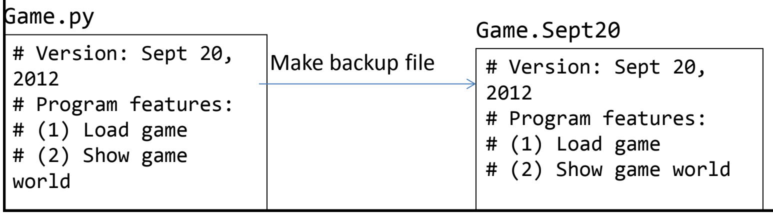
The number sign ‘#” flags the translator that the remainder of the line is documentation.

- **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.
```

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.



Game.py

```
# Version: Sept 20,
2012
# Program features:
# (1) Load game
# (2) Show game
world
```

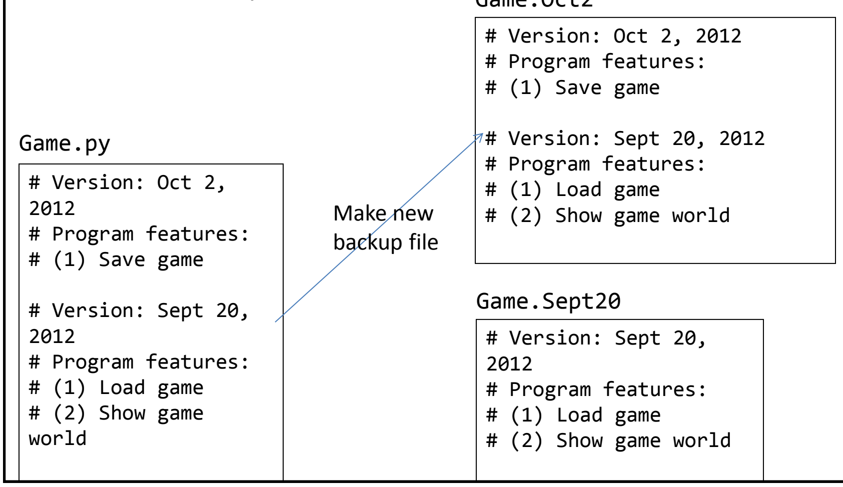
Make backup file

Game.Sept20

```
# Version: Sept 20,
2012
# Program features:
# (1) Load game
# (2) Show game world
```

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.



Game.Oct2

```
# Version: Oct 2, 2012
# Program features:
# (1) Save game

# Version: Sept 20, 2012
# Program features:
# (1) Load game
# (2) Show game world
```

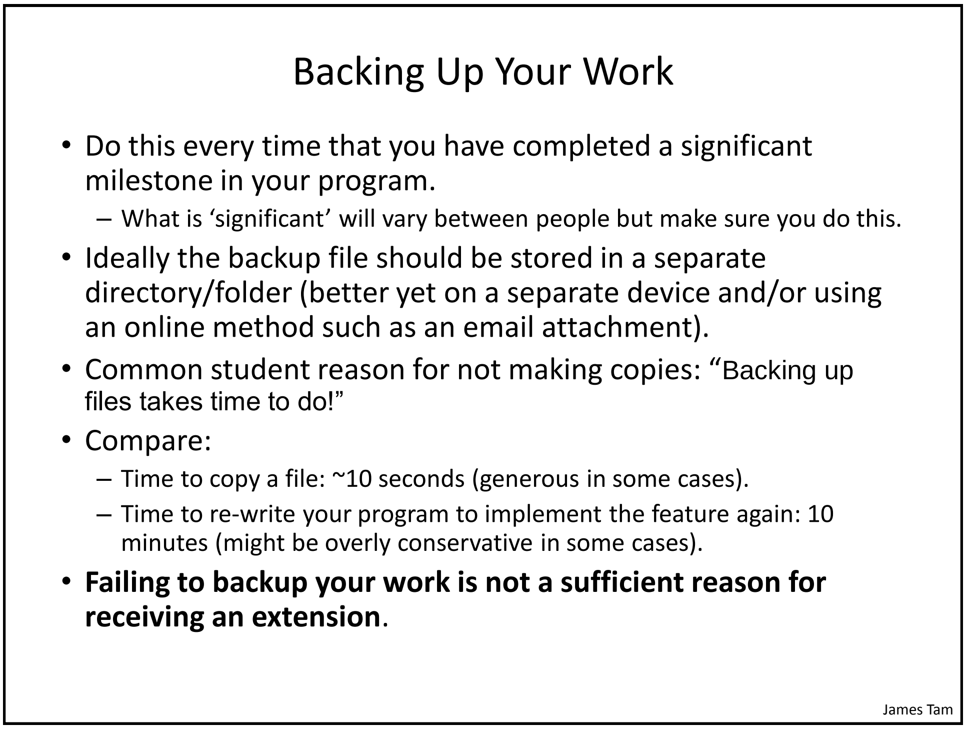
Game.py

```
# Version: Oct 2,
2012
# Program features:
# (1) Save game

# Version: Sept 20,
2012
# Program features:
# (1) Load game
# (2) Show game
world
```

Make new backup file

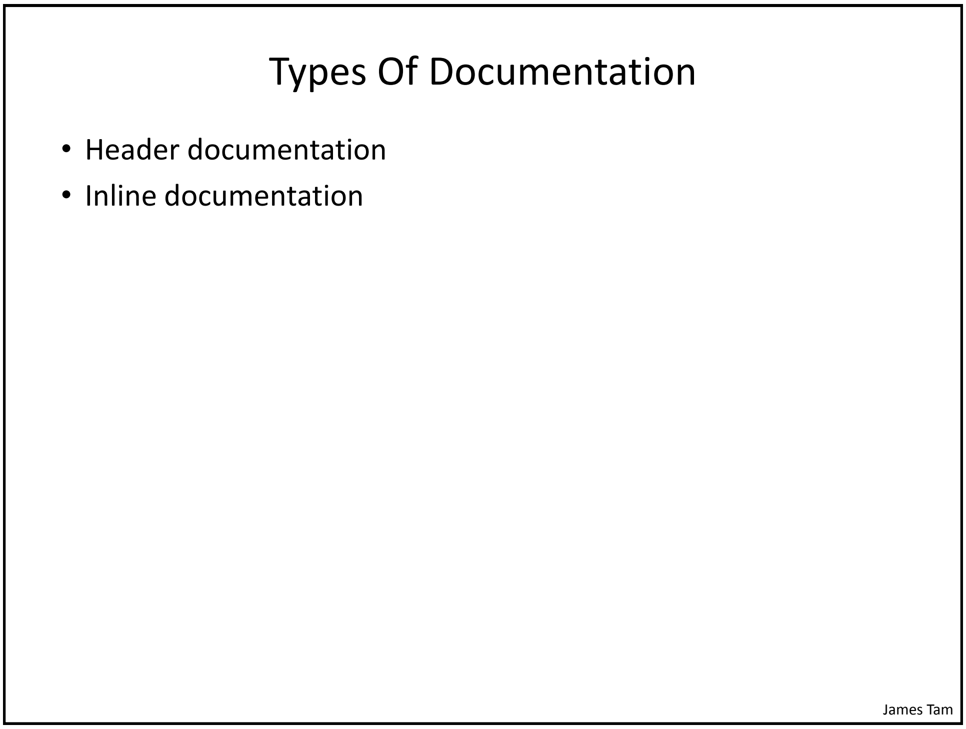
Game.Sept20

```
# Version: Sept 20,
2012
# Program features:
# (1) Load game
# (2) Show game world
```



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 the 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**.



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 (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>
```

Section Summary: Documentation

- What is program documentation
- What sort of documentation should be written for your programs
- How program documentation ties into program versioning and backups

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 (you can assume that you cannot use a function unless: (1) it’s extremely common e.g., input and output) (2) it’s explicitly allowed
  - Read the requirements specific to each assignment
  - When in doubt don’t use the pre-created code either ask or don’t use it and write the code yourself. (**If you end up using a pre-created function rather than writing the code yourself you could receive no credit**).

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:

```
5 = num
```

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

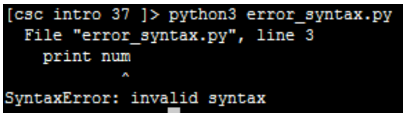
1. Syntax/ Translation Errors (2)

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

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 e.g., print(“hello)
- Using variables before they’ve been named (allocated in memory).
- Program name: `error_syntax.py`

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



```
[csc intro 37 ]> python3 error_syntax.py
  File "error_syntax.py", line 3
    print num
            ^
SyntaxError: invalid syntax
```

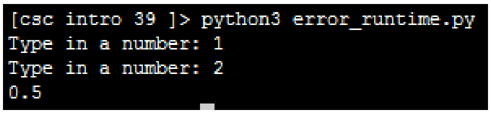
2. Runtime Errors

- 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.
  - We will talk about other run time errors later.

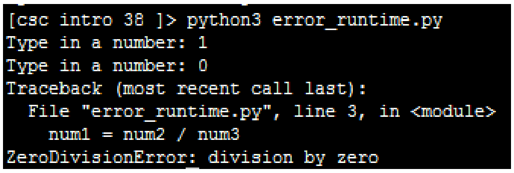
2. Runtime Error[1]: An Example

- Program name: `error_runtime.py`

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



```
[csc intro 39 ]> python3 error_runtime.py
Type in a number: 1
Type in a number: 2
0.5
```



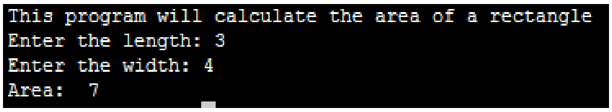
```
[csc intro 38 ]> python3 error_runtime.py
Type in a number: 1
Type in a number: 0
Traceback (most recent call last):
  File "error_runtime.py", line 3, in <module>
    num1 = num2 / num3
ZeroDivisionError: division by zero
```

[1] When 'num3' contains zero

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)
```



```
This program will calculate the area of a rectangle
Enter the length: 3
Enter the width: 4
Area:  7
```

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

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

Section Summary: The 3 Error Types

- What are different categories of errors
- What is the difference between the categories of errors and being able to identify examples of each

James Tam

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
```

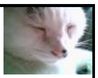
James Tam

Layout And Formatting

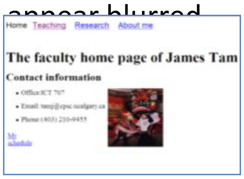
- Good layout and formatting makes it easier to read and understand your program.
- This is crucial for all but small (unrealistic) programs.

James Tam

The Squint Test: A Tool For Evaluating Layout



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



Home Teaching Research About me

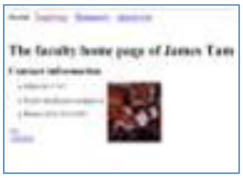
The faculty home page of James Tam

Contact information

- Office:ICT 707
- Email: tamj@cpsc.ucalgary.ca
- Phone:(403) 210-9455

My schedule

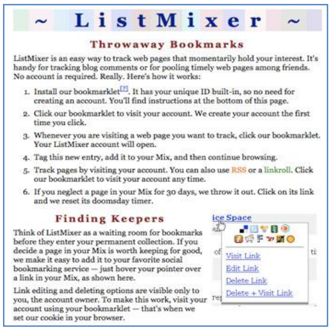
Original webpage



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

A Webpage That Fails The Squint Test



~ ListMixer ~

Throwaway Bookmarks

ListMixer is an easy way to track web pages that momentarily hold your interest. It's handy for tracking blog comments or for pooling timely web pages among friends. No account is required. Really. Here's how it works:

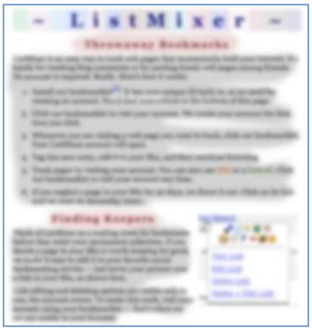
1. Install our bookmarklet[?]. It has your unique ID built-in, so no need for creating an account. You'll find instructions at the bottom of this page.
2. Click our bookmarklet to visit your account. We create your account the first time you click.
3. Whenever you are visiting a web page you want to track, click our bookmarklet. Your ListMixer account will open.
4. Tag this new entry, add it to your Mix, and then continue browsing.
5. Track pages by visiting your account. You can also use RSS or a linkroll. Click our bookmarklet to visit your account any time.
6. If you neglect a page in your Mix for 30 days, we throw it out. Click on its link and we reset its doomsday timer.

Finding Keepers

Think of ListMixer as a waiting room for bookmarks before they enter your permanent collection. If you decide a page in your Mix is worth keeping for good, we make it easy to add it to your favorite social bookmarking service — just hover your pointer over a link in your Mix, as shown here.

Link editing and deleting options are visible only to you, the account owner. To make this work, visit your account using your bookmarklet — that's when we set our cookie in your browser.

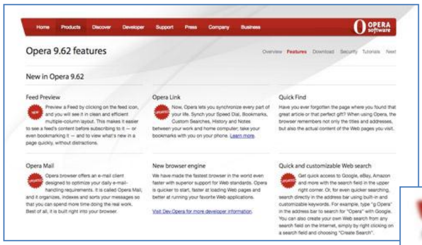
Original webpage



Blurred version

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

A Webpage With Better Squint Test Results



Opera 9.62 features

New in Opera 9.62

Feed Preview

Preview a Feed by clicking on the feed icon, and you will see it in clean and efficient multiple-column layout. This makes it easier to see a feed's content before subscribing to it — or even bookmarking it — and to view what's new in a page quickly, without distractions.

Opera Link

Now, Opera lets you synchronize every part of your life. Synch your Speed Dial, Bookmarks, Custom Searches, History and Notes between your work and home computer; take your bookmarks with you on your phone. Learn more.

Quick Find

Have you ever forgotten the page where you found that great article or that perfect gift? When using Opera, the browser remembers not only the titles and addresses, but also the actual content of the Web pages you visit.

Opera Mail

Opera browser offers an e-mail client designed to optimize your daily e-mail-handling requirements. It is called Opera Mail, and it organizes, indexes and sorts your messages so that you can spend more time doing the real work. Best of all, it is built right into your browser.

New browser engine

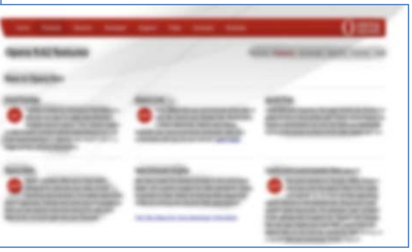
We have made the fastest browser in the world even faster with superior support for Web standards. Opera is quicker to start, faster at loading Web pages and better at running your favorite Web applications.

Visit Dev.Opera for more developer information.

Quick and customizable Web search

Get quick access to Google, eBay, Amazon and more with the search field in the upper right corner. Or, for even quicker searching, search directly in the address bar using built-in and customizable keywords. For example, type "g Opera" in the address bar to search for "Opera" with Google. You can also create your own Web search from any search field on the Internet, simply by right clicking on a search field and choosing "Create Search".

Original webpage



Blurred version

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

James Tam

Section Summary: Layout And Formatting

- Why is layout and formatting of programs important, how to do it

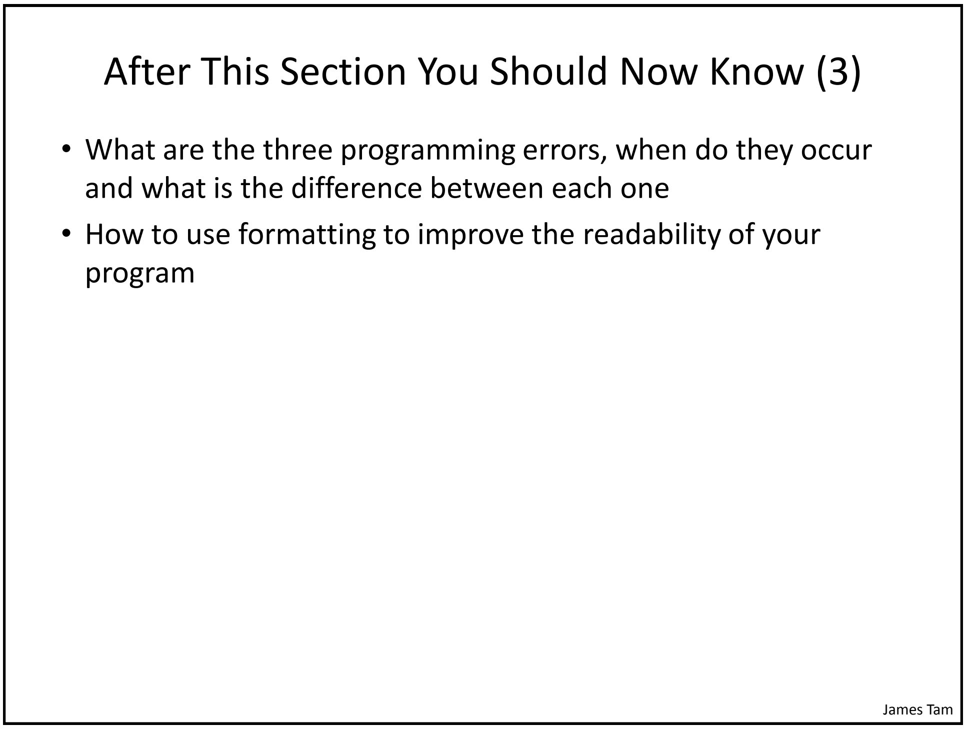
James Tam

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

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



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
- How to use formatting to improve the readability of your program