

Getting Started With Python Programming

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

James Tam

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 © Monty Python

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

James Tam

Online Help

- Getting Python (*get version 3.X* and not version 2.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>
 - <http://docs.python.org/using/unix.html>
 - <http://docs.python.org/using/mac.html>
- (Alternate to installing Python at home – SAFER APPROACH).
 - 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>)

James Tam

Online Help

- (If you have installed Python on your own computer and still can't get 'Python' to run – it works although it's a 'inelegant' solution)
 - Note where you installed Python (folder or directory)
 - Create and run your Python programs from this location.
- Explanation of concepts (for beginners: along with examples to illustrate)
 - <http://docs.python.org/tutorial/>
- General documentation (more advanced):
 - <http://www.python.org/doc/>

James Tam

The Process Of Creating A Computer Program

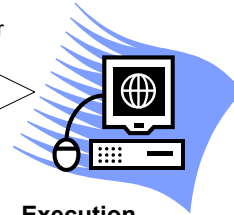
Translation



- 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/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

James Tam

Location Of My Online Examples

- Finding them via the WWW:
 - URL: <http://pages.cpsc.ucalgary.ca/~tamj/231/examples/>
- Finding them on UNIX when you are logged onto a computer in the lab:
 - 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

An Example Python Program

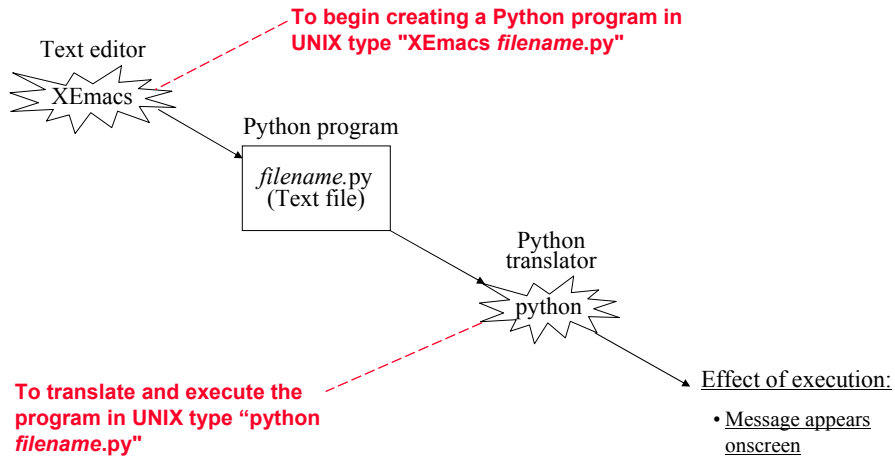
- Program name: small.py

Filename: small.py

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

James Tam

Creating, Translating And Executing Python Programs (CPSC Network¹)



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

James Tam

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

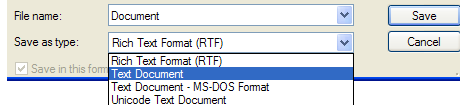
- **Creating the program in an editor:** Type "emacs/xemacs small.py"
 - A file called "small.py" will be created in your UNIX account.
- **Translating and running the program:** Type "python small.py"
 - Make sure you type this command in the location (i.e., same directory) where the Python program (small.py) is located.

James Tam

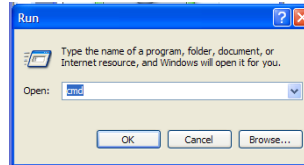
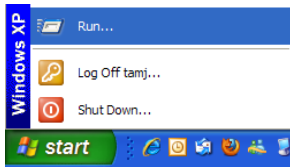
Creating Programs: Other Operating Systems

- 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.



James Tam

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 ('inelegant but works')

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\tanj>python small.py
python: can't open file 'small.py': [Errno 2] No such file or directory
C:\Documents and Settings\tanj>
```

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)

James Tam

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")
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.
 - *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> = <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")



Picture from Computers in your future by Pfaffenberger B

James Tam

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!)

James Tam

Key Words In Python¹

| | | | | |
|----------|---------|--------|--------|-------|
| 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

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).
- 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

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

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

Why Use Named 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!)**

James Tam

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.

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 (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

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

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.

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:
 - INCH_CENTIMETER_RATIO = 2.54
 - height = height * INCH_CENTIMETER_RATIO
- Example 2:
 - Calories used = (10 x weight) + (6.25 x height) - [(5 x age) - 161]

James Tam

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

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

Output: Problems

- Sometimes Python automatically adds 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

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

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

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

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

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

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

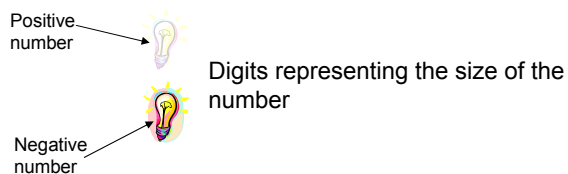


James Tam

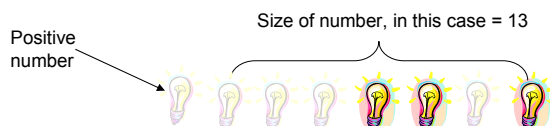
Storing Integer Information

- 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:

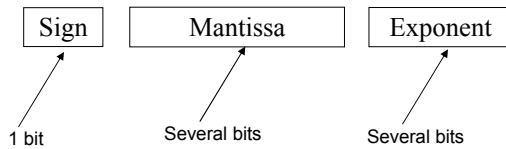


- Previous example



James Tam

Storing Real Numbers In The Form Of A Float



- 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

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

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

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

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

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

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))
```

James Tam

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))
```

James Tam

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

Some Formatting Effects Using Descriptor Codes

•Format:

`%<width>1.<precision>2<type of information>`

•Examples:

- Program name: formatting2.py

```
num = 12.55
print ("%4.1f" %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

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

```
***
C
O
M
P
I
L
E
R
***
```

From Python Programming (2nd Edition) by Michael Dawson

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. |
| \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. |

James Tam

Escape Codes (2)

- Program name: formatting4.py

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

James Tam

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 its limitations etc.

James Tam

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 whole do e.g., tax program.
 - What are the **specific features** of the program e.g., it calculates personal or small business tax.
 - What are its **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

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

Types Of Documentation

- Header documentation
- Inline documentation

James Tam

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

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>
```

James Tam

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>

James Tam

Types Of Programming Errors


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

James Tam

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.

James Tam

1. Syntax/ Translation Errors (2)

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

James Tam

1. Some Common Syntax Errors

- Miss-spelling names of keywords
- e.g., 'print' 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

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

2. Runtime Error¹: 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

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

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

After This Section You Should Now Know (2)

- Output:
 - How to display messages that are a constant string or the value of 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

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