Getting Started With Python Programming: Part I

- Creating/running computer programs
- Variables
- Displaying information (output)
- Getting information from the user (input)
- Common mathematical operators

Reminder!

- These course notes are mandatory
  - Get them before class and go over them before attending
- (If all else fails then look through them afterwards – at the very least to see what concepts/topics you are responsible for knowing).
  - It's the *first* step you should complete if you've missed lecture and need to catch up.
  - (The second step is to get the in class notes of a classmate).
  - After going through these notes the third step is to ask us for help in filling in any conceptual gaps.

James Tam

Python History

- Developed in the early 1990s by Guido van 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

Online Help: Official Python Site

- *Basic explanation* of concepts (for beginners: along with examples to illustrate)
  - http://docs.python.org/py3k/tutorial/index.html
  - (Skip the notes on the interactive mode for now – it's where you don't save the program which leaves you nothing to submit for assignments).
  - For this course you need to create a python program in a file and then run the program defined in the file.
- Style guidelines (note they are very general and don't provide a lot of specific details but this domain is owned by the same foundation by the "Python software foundation"):
  - https://peps.python.org/pep-0008/

James Tam

Creating A Computer Program

1. A programmer writes the instructions of the program in high level (human can read and understand) language.
   – Examples: C, C++, java, python

```
# Details later this term
list = [1,2,'a']
for element in list
    print(element)
```

2. The program must be created and saved using a text editor (e.g. Notepad, WordPad or the editor that comes with python IDLE).

The program is then translated into machine language (**the only form that the computer can understand**).

```
# Details in 2nd year
10000001
10010100 10000100
10000001 01010100
```

James Tam

Program Translation (Python)

- Your python programs are interpreted.
- This means that they are translated one instruction at a time into machine language so each machine language instruction is run one at a time.

James Tam

Location Of My Online Examples

- For this semester you can find them in D2L under: `Content->L02`
- Then look under the appropriately named folder which is listed by date and topic.
- Alternatively you can find them by looking under the "main grid" of the course website (look for the 'examples' link):
  - https://cspages.ucalgary.ca/~tam/2025/217F/#Main_grid:_Course_topics,_course_schedule

James Tam

The First Python Program

Name of the full example: `1small.py`

Filename: `1small.py`

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

James Tam

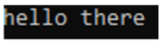
Step-By-Step Walkthrough Creating & Running A Program

- These notes may be helpful in recalling the steps (where/what to click) needed to create/run a python program.
- Two editors will be covered:
  - PyCharm
  - IDLE
- Link:
  - https://cspages.ucalgary.ca/~tam/2025/217F/#Lecture_week_1
  - (Look for the text "creating/running python programs)

James Tam

Displaying Output Using The `Print()` Function:

- This function takes zero or more arguments (inputs)
  - Multiple arguments are separated with commas (extra blank appears in the output). `print("hello", "there")`

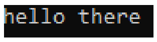


```
hello there
```

  - `print()` will display all the arguments followed by a blank line (the cursor is automatically moved to the next line).
  - Zero arguments just displays a blank line

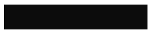
- **Name of the full example**: `2outputExtras.py`

```
print("hello", "there")
```

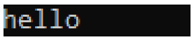


```
hello there
```

```
print()
```

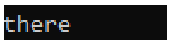


```
print("hello")
```



```
hello
```

```
print("there")
```



```
there
```

James Tam

Variables

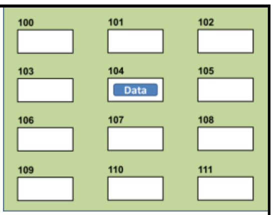


Image curtesy of James Tam

- 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 i.e., it's temporary
- Some types of information which can be stored in variables include:
  - integer (whole number storage only) e.g. `age = 37`
  - floating point (fractional) e.g. `height = 68.5`
  - strings (character information - essentially any characters you can type and more) e.g. `fightingName = "TamJet"` (enclosed in double quotes – do not use single quotes as other languages use this type of quote)
  - The type of information on the RHS of the **assignment operator** (**=**) determines the type of information stored.

slide 11

James Tam

Updating Variables

- Knowing that each assignment overwrites previously values is important as the updating of variables is a common operation.
- Example 1: incrementing (increasing by one) operation

```
count = count + 1     (increase count by 1)
```

- Example 2: stepping through multiples of an integer.

```
multiple = 5
multiple = multiple * 5     (multiple contains 25)
multiple = multiple * 5     (multiple contains 125)
```

James Tam

The Assignment Operator: =

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

```
y = 3 (what is stored in 'y' at this point)
x = y (what is stored in 'x','y' at this point)
y = 6 (what is stored in 'x','y' at this point)
```

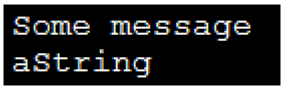
- What is the end result? How was this derived (what are the intermediate results)
  - Recall: Putting another piece of information at an existing location *overwrites* previous information.
- **Name of the full example (shows output):** `3assignment.py`
  - Quick tip after using the assignment operator: to show what a variable currently contains put the name of the variable without quotes inside the round brackets for the print function e.g.,
    - num = 888
    - `print(num)`

James Tam

`Print("… ")` Vs. `Print(<name>)`

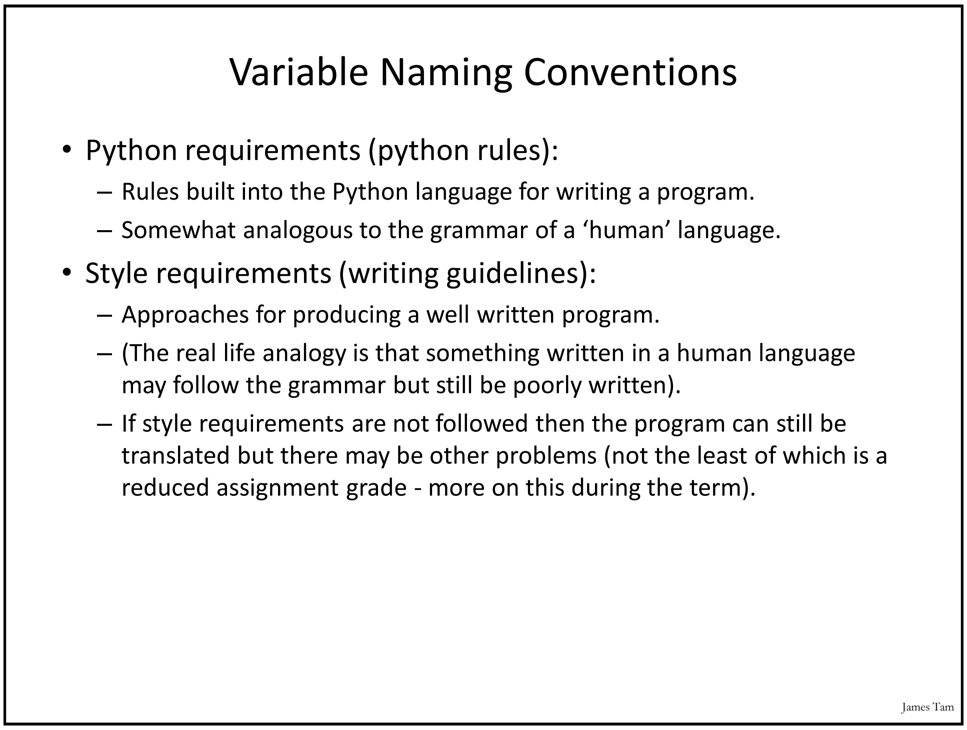
- Enclosing the value in brackets **with quotes** means the value in between the quotes will be literally displayed onscreen.
- **Excluding the quotes** will display the contents of a memory location.
- **Name of the full example:** `4outputQuotes.py`

```
aString = "Some message"
print(aString)
print("aString")
```



```
Some message
aString
```

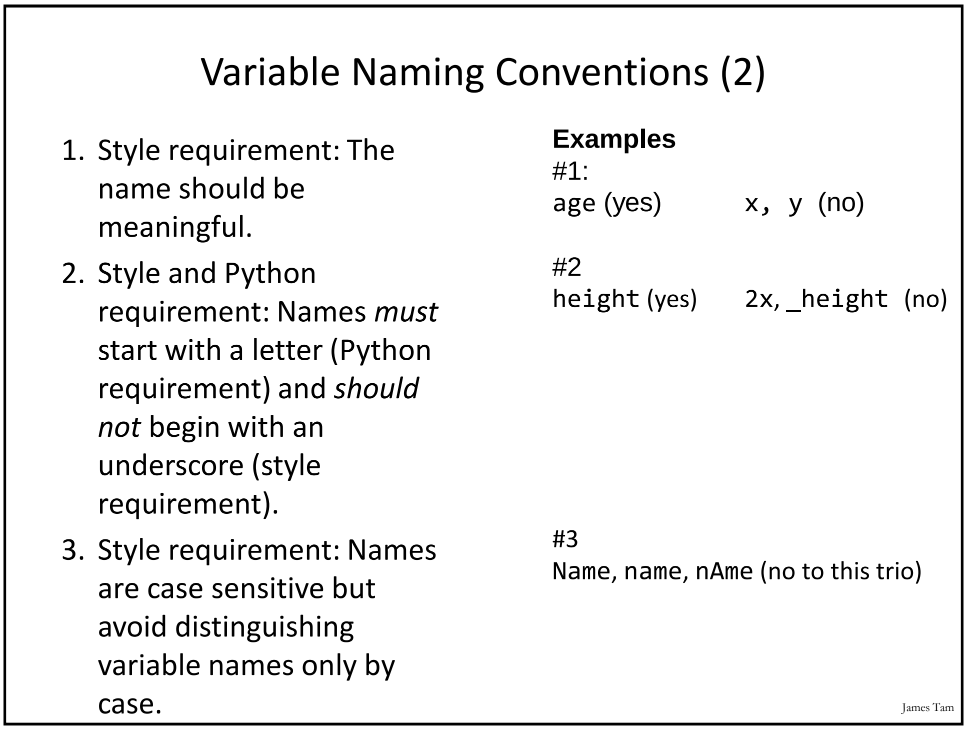
James Tam



Variable Naming Conventions

- Python requirements (python rules):
  - Rules built into the Python language for writing a program.
  - Somewhat analogous to the grammar of a ‘human’ language.
- Style requirements (writing guidelines):
  - 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 (not the least of which is a reduced assignment grade - more on this during the term).

James Tam



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)

James Tam

Variable Naming Conventions (3)

4. Style requirement: Variable names should generally be all lower case (see the next point for the exception).
5. Style requirement: For 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 e.g. print()

Examples

#4:

`age, height, weight`	(yes)
`Age, HEIGHT`	(no)

#5 ("camel case", "snake case")

`firstName`, `last_name`

(yes to either approach but keep in mind that using the underscore is the convention in python)

James Tam

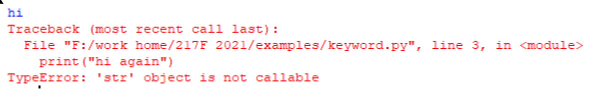
Some Key Words In Python

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

Program instructions

```
print("hi")
print = "bye"
print("hi again")
```

Result of running the program



```
hi
Traceback (most recent call last):
  File "F:/work home/217F 2021/examples/keyword.py", line 3, in <module>
    print("hi again")
TypeError: 'str' object is not callable
```

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.
  - This means changes should not occur because of style reasons rather than because Python prevents the 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
- Variables are all lower case to provide a clear contrast.
- 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 named constant and not to change it.

James Tam

Why Use **Named Constants**

1. They make your program easier to read and understand

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

Avoid unnamed constants whenever possible!

Vs.

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

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

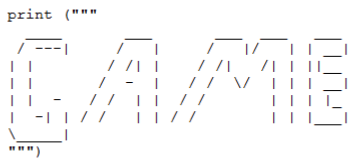
Purpose Of **Named Constants** (3)

```
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

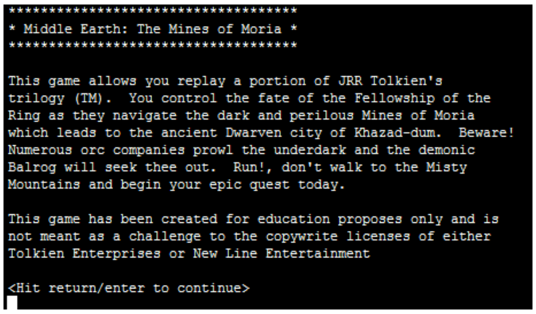
Triple Quoted Output

- 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.
- **Name of the full example**: 5tripleQuotes.py



```
print ("""
  ____       ___       ___  ___   ___
 / ---|     /   |     /   |/   | |  _|
| |        / /| |    / /|   /| | | |__
| |       /  - |    / /  \/  | | |  _|
| |  -   / /  | |  / /       | | |  |_
|  -| | / /   | | / /        | | |___|
\_____|
""")
```

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>
```

From a CPSC 231 assignment (image courtesy of James Tam)

James Tam

Arithmetic **Operators**

- **Name of the full example:** `6operators.py`

Operator	Description	Example	
=	Assignment	num = 7	
+	Addition	num = 2 + 2	
-	Subtraction	num = 6 - 4	
*	Multiplication	num = 5 * 4	
/	Division (real number)	num = 9 / 2	4.5
//	Integer division	num = 9 // 2	4
%	Modulo (remainder)	num = 9 % 2	1
**	Exponent	num = 9 ** 2	81

James Tam

Order Of Operation: "Same As Math"/ "B E D-M A-S"

- 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.
  - **Name of the full example:** `7order.py`

1st	`()`	Brackets (inner before outer)
2nd	`**`	Exponent
3rd	`*, /, //, %`	Multiplication, division, modulo
4th	`+, -`	Addition, subtraction
5th	`=`	Assignment

Example

```
num = 3 * 2 ** 3

Vs.
num = (3 * 2) ** 3
```

James Tam

Basic Input

- The computer program getting *string information* from the user.
- Strings cannot be used for calculations (information for getting numeric input will be covered later, early hint: you need to convert it).

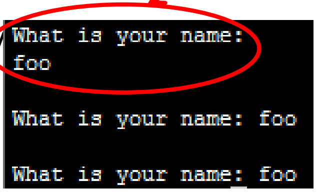
- **Format:**

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

- **Name of the full example:** `8input.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
```

James Tam

Conceptual Recap

- **New term:** What you have seen so far are examples of **programming statements** in python.
  - Statements are instructions to be executed.
  - Since the computer can only execute machine language instructions the python statements must be translated prior to execution.
  - Reminder: any errors in forming valid python instructions (dictated by the syntax of python) will not allow that instruction to execute as no translation occurs.

James Tam

Program Documentation (“Comments”)

- *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**.
- Unlike program instructions (e.g. calling functions such as `print` or performing a calculation), **documentation does not execute**.
- 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

Program Documentation (2)

- **Format (single line documentation):**

```
# <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.
```

James Tam

Program Documentation (3)

- **Format (multiline documentation):**

```
""" <Start of documentation>
 ...
 <End of 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

What To Write In Documentation

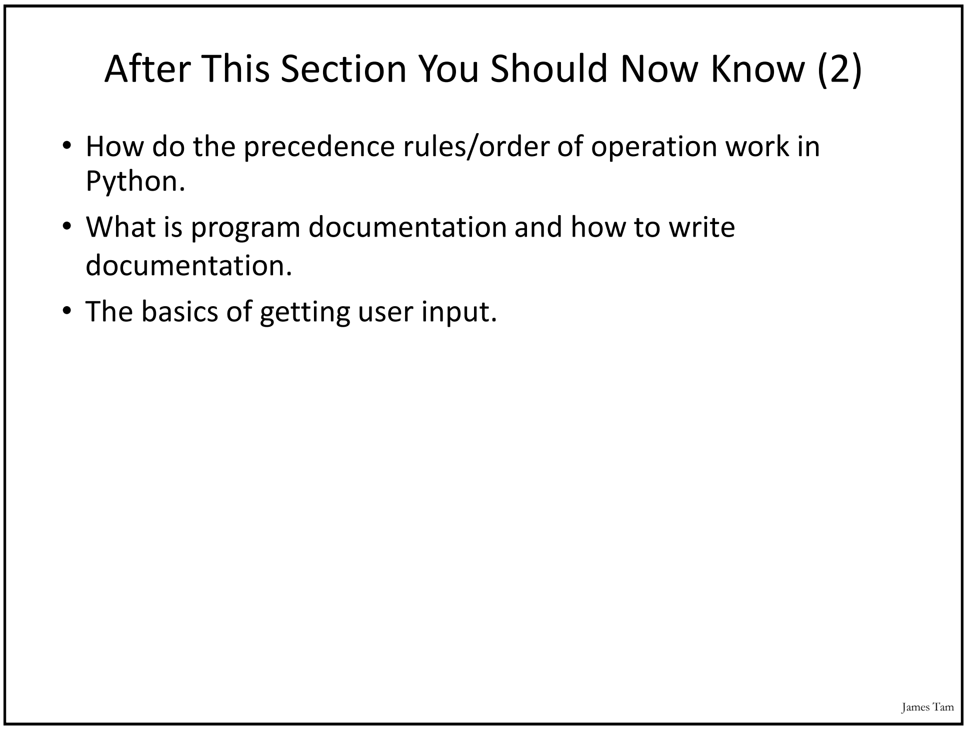
- As needed you will be provided with documentation required for this course (likely in the assignment descriptions).
- Here's some general things you can include.
  - **The author** of the program (or for a particular part of a program).
  - **What does** the program as a while do e.g., calculate common math functions.
  - What are the **specific features** of the program e.g., list the specific operations
  - What are it's **limitations** e.g., cannot perform a division by zero, negative angles not allowed for trigonometric functions.
  - What is the **version** of the program.
    - If you don't use numbers for the different versions of your program then simply use dates.
    - You can then get into the good habit of backing up (or submitting into D2L) each version.

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.
  - How information is stored differently with different types of variables.
- Named constants:
  - What are named constants and how they differ from regular variables.
  - What are the benefits of using a named constant vs. unnamed constant.
- 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/why use triple quoted output.
- What are the Python operators for common mathematical operations.

James Tam



After This Section You Should Now Know (2)

- How do the precedence rules/order of operation work in Python.
- What is program documentation and how to write documentation.
- The basics of getting user input.

James Tam