

Getting Started With Python Programming: Part III

- Style: variable naming conventions
- Named constants
- Documenting programs
- Prewritten python functions
- Common programming errors
- Programming style: layout and formatting of your program

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 (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 (2)

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 (see next slide).

Examples

#4:

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

#5

firstName, last_name
(yes to either approach)

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

Variable Naming Conventions: Bottom Line

- Both Python and style requirements should be followed when creating your variables.

James Tam

Extra Practice

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

James Tam

Section Summary: Variables

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

James Tam

Reminder: Variables

- By convention variable names are all lower case
- The exception is long (multi-word) names
- As the name implies their **contents can change** as a program runs e.g.,

```
income = 300000
```

```
income = income + interest
```

```
Income = income + bonuses
```

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
- 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 named constant and not to change it.

James Tam

Variables Vs. Constants: Case Study

- First graded assignment: when deciding if a memory location (an identifier) should be treated as a constant or a variable ask yourself:
 - During the course of a semester should the contents of that identifier have the ability to change?
 - Full assignment 1 grade?
 - One of the exam grades?
 - The weight of Full Assignment 2?
 - The weight of the midterm?

James Tam

Why Use **Named Constants**

1. They make your program easier to read and understand

NO

```
populationChange = (0.1758 - 0.1257) * currentPopulation
```

Vs.

**Avoid unnamed constants
whenever possible!**

#YES

```
BIRTH_RATE = 17.58
```

```
MORTALITY_RATE = 0.1257
```

```
currentPopulation = 1000000
```

```
populationChange = (BIRTH_RATE - MORTALITY_RATE) *  
    currentPopulation
```

James Tam

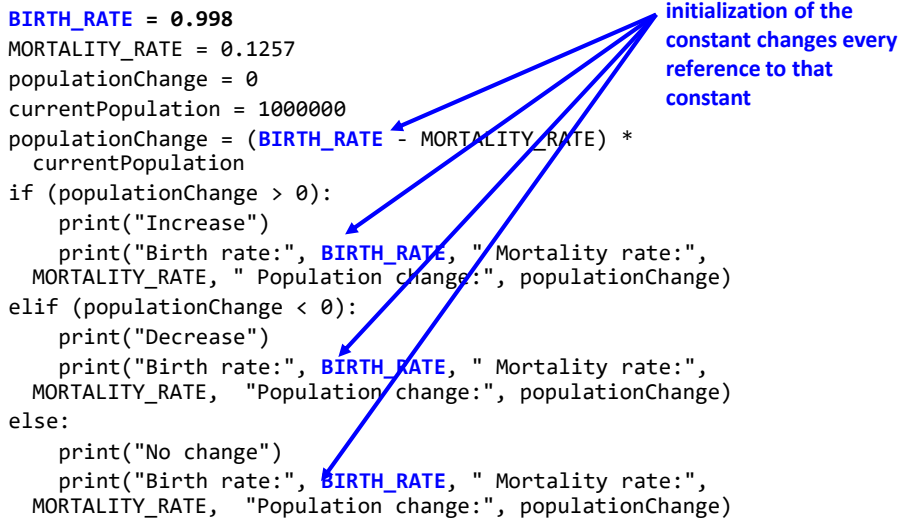
Why Use Named Constants (2)

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.

James Tam

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

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 define and use a named constant.
- **Example 1** (easy to provide self explanatory constant name)


```

INCH_CM_RATIO = 2.54
heightInches = heightCM * INCH_CM_RATIO

```
- **Example 2** (providing self explanatory names for the constants is difficult)


```

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

```

James Tam

Named Constants: A Final Example

Correct/incorrect use of named constants can affect your assignment grade

- Which of the following programs is more self explanatory ("self documenting" code)?
 - (You will learn how the 'IF' works in the branching/decisions making lectures).

– **Example #1 (less clear):**

```
gameStatus = 1
silverLockPosition = 2
goldLockPosition = 0
if ((silverLockPosition == 1) and (goldLockPosition == 0)):
    gameStatus = 2
```

– **Approach #2 (more explicit, better):**

```
WON = 2
LEFT = 0
RIGHT = 1
CENTER = 2
If ((silverLockPosition == RIGHT) and (goldLockPosition == LEFT)):
    gameStatus = WON
```

James Tam

Extra Practice

- Provide a formula where it would be appropriate to use named constants (should be easy).
- Provide a formula where unnamed constants (i.e., named constant used instead of named constants) may be acceptable (may be trickier).
- Search for formulas in science or engineering sites online if you can't think of any formulas.

James Tam

Section Summary: Named Constants

- What is a named constant
 - How does it differ from a variable
 - How does it differ from an unnamed constant
 - What are some reasons for using named constants
- Naming conventions for named constants

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)

- Doesn't contain instructions for the computer to execute.
- Not translated into machine language.
- Consists of information for the reader of the program:
 - **The author** of the program (or for a particular part of a program).
 - **What does** the program as a whole do e.g., calculate taxes.
 - 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 simply use dates (tie versions with program features – more on this in a moment "Program versioning and backups").

James Tam

Program Documentation (3)

- **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 (4)

- **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

Assignment Documentation Requirements

- Information about you: author contact information (full name, student identification number, tutorial number that you are registered in).
- Other information to document:
 - Program version
 - List of features in the assignment description that your program implemented for each version (paraphrase or even copy-pasting of requirements is acceptable).
 - Any weaknesses or limitation of your program (e.g. 1: program crashes if a non-numeric value is entered when a number is expected, e.g. 2: program cannot calculate a quotient if the user enters denominator of zero).
 - See the requirements of the specific assignment for any additional details.

James Tam

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
```

James Tam

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: 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.Oct2

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

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

Game.Sept20

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

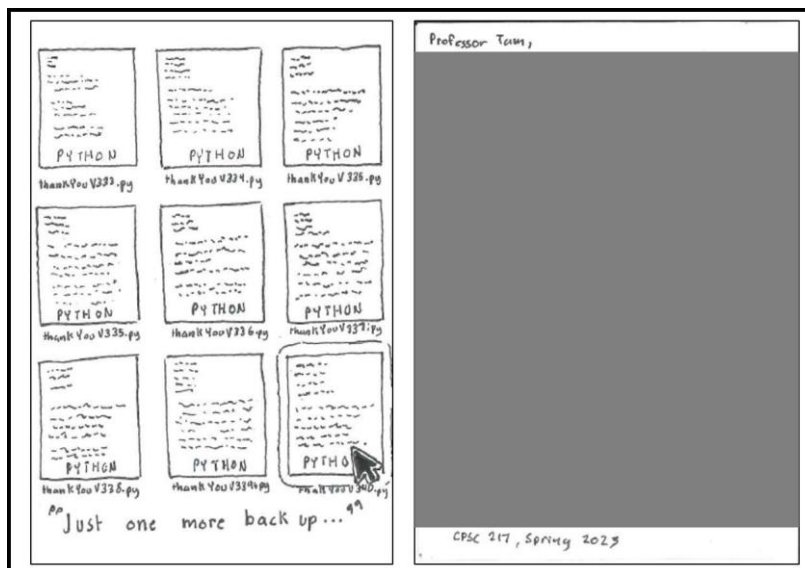
James Tam

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 periodically.
- 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 or 'cloud' storage).
- Common student reason for not making copies: "Backing up files takes time!"
- 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.**

James Tam

Backups: On A Positive Note



James Tam

Over-Documenting A Program

- Except for very small programs documentation should be included
- However, it is *possible* to over-document a program
- (Stating the obvious)
`num = num + 1 #Variable num increased by one`
- (Documentation of the last row in a list may be a good reminder)
`lastRow = SIZE - 1 #Row numbering begins at zero`

Example: there are 3 rows in a list (SIZE = 3)

- First row = 0
- Second row = 1
- Third (and last) row = 2 (equals 3-1 = 2)

James Tam

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

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.
 - <https://docs.python.org/3/library/functions.html>
 - Note: some assignments may have specific instructions which list functions you are allowed to use (**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**).

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:
 - My cat sleeps the sofa.
- Python statements are structured by the *syntax* of Python:

5 = num

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

Syntactically incorrect: the left hand side of an assignment statement cannot be a literal (unnamed) constant (or variable names cannot begin with a number)

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 e.g., `print("hello)`
- Using variables before they've been named (allocated in memory).
- **Name of the full example:** 17error_syntax.py

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

```
Traceback (most recent call last):
  File "syntax.py", line 1, in <module>
    print(num)
NameError: name 'num' is not defined
```

James Tam

2. Runtime Errors

"My computer crashed!"

- 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.

James Tam

2. Runtime Error¹: An Example

- **Name of the full example:** 18error_runtime.py

```
num2 = int(input("Type in a number: "))
num3 = int(input("Type in a number: "))
num1 = num2 / num3 # When zero is entered
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

James Tam

3. Logic Errors

Software "bugs"

- 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).
- **Name of the full example:** 19error_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
```

James Tam

Some Additional Examples Of Errors

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

James Tam

Practice Exercise

- (This one will be an ongoing task).
- As you write your programs, classify the type of errors that you encounter 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 computer 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 for 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: Example

```
# Creating reference to grid
aGrid = []

# Creating the grid data
for r in range (0,noRows,1):
    aGrid.append ([])
    for c in range (0,noColumns,1):
        aGrid[r].append("*")

# Displaying the grid
for r in range (0,noRows,1):
    for c in range (0,noColumns,1):
        print(aGrid[r][c], end="")
    print()
```

James Tam

Section Summary: Layout And Formatting

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

James Tam

Extra: In Case You're Interested

- Different languages may have unique style guides
- Here a style guide for Python (FYI not as well developed as for other languages):
 - <https://www.python.org/doc/essays/styleguide/>
 - <https://peps.python.org/pep-0008/> (sub-link of the above link)
- Oracle's Java style guide:
 - <https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>
 - <https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html>
- Microsoft's style guide (specifies convention for writing in English and it's included just to demonstrate how detailed style guides can be):
 - <https://learn.microsoft.com/en-us/style-guide/welcome/>

James Tam

After This Section You Should Now Know

- Conventions for naming 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
- What is program documentation and what are some common things that are included in program documentation
- The existence of prewritten Python functions and how to find descriptions of them
- What are the three programming errors, when do they occur and what is the difference between each one
- The importance of formatting program code and some guidelines for formatting.

James Tam