Road Map

Algorithm Design and Analysis

- Week 1: Intro, Counting, Programming
- Week 2: Programming
- Week 3: Complexity
- Week 4: Iteration & Decomposition
- Week 5: Recursion

Here now

Fundamental Data Structures

(Weeks 6 - 10)

Computational Intractability and Heuristic Reasoning

(Weeks 11 - 13)

School of
Information Systems

SMU
SINGAPORE MANAGEMENT UNIVERSITY

COR-IS1702: Computational Thinking
Week 1 (programming): Python Workbench

2020/21 Term 1

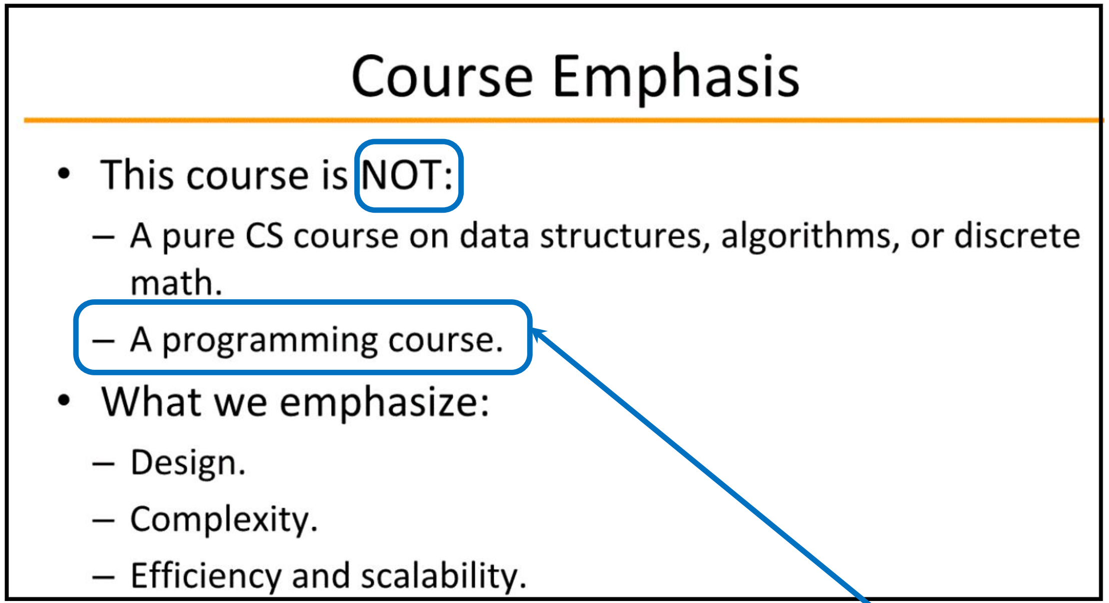
Contact Details

Contact your instructor anytime for:

- Help on lab
- Enquiries about projects
- Anything related to programming

Section	Instructor	Contact
G5 Thu 12:00 - 15:15	MOK Heng Ngee	hnmok@smu.edu.sg Telegram: @Mokkie
G6 Wed 815 - 1130	LEE Fiona	fionalee@smu.edu.sg Telegram: @fionaleeyy
G7 Fri 815 - 1130	LEE Fiona	fionalee@smu.edu.sg Telegram: @fionaleeyy

Recall.... Introduction slide:



Course Emphasis

- This course is NOT:
  - A pure CS course on data structures, algorithms, or discrete math.
  - A programming course.
- What we emphasize:
  - Design.
  - Complexity.
  - Efficiency and scalability.

But…. you will need to "articulate" your algorithms in the form of a program for your **projects**!
Which means… you have to learn basic programming anyway :-)

You are allowed to give your solutions as "pseudo-code" for the **quizzes/exams** though.

Set up Python

- You will need Python 3 for this course
  - MacOS comes with Python 2 by default; you need to install Python 3
  - Windows does not come with Python. Install it.
- How to install Python:
  - see http://ix.cs.uoregon.edu/~conery/eic/python/installation.html (go to section on "Install Python and Tcl/Tk")
  - Tcl/Tk is only required for graphics & animation. You need Tcl/Tk in order to view some visualizations when running some code samples in later units. Otherwise, it is not necessary to install Tcl/Tk

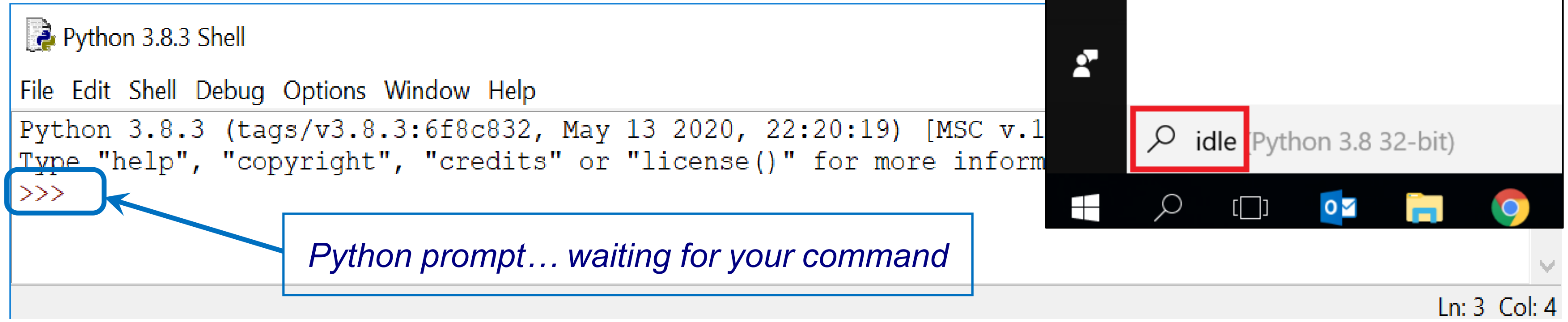
…Have you set up Python Correctly?

IDLE is installed by default with any Python installation.

- This is a shell for entering Python commands interactively
- Stands for **I**ntegrated **D**eve**L**opement **E**nvironment

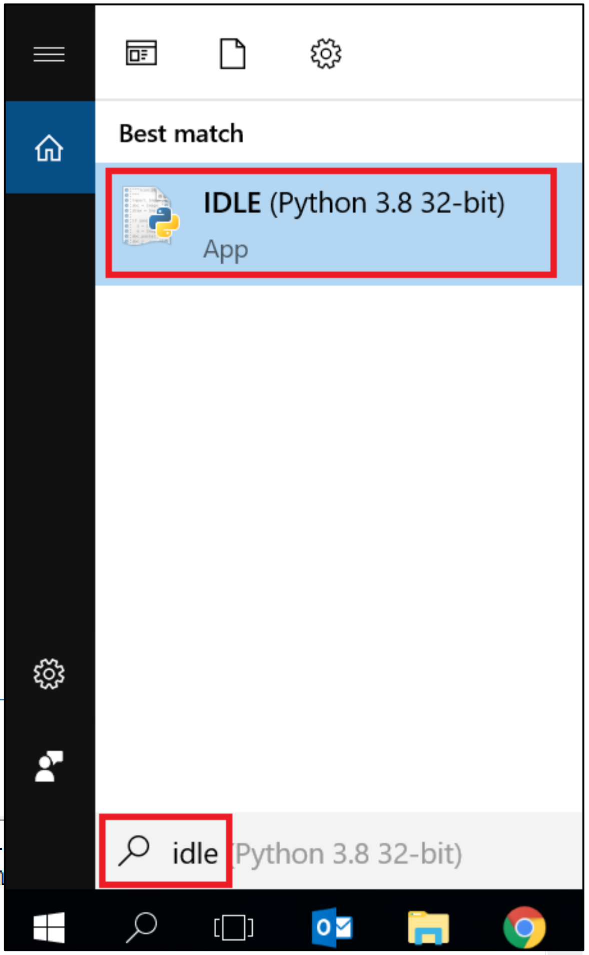
Windows: click START and type "IDLE".

MacOS: Applications-->Python3.7-->IDLE



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1
Type "help", "copyright", "credits" or "license()" for more inform
>>>
```

Python prompt… waiting for your command

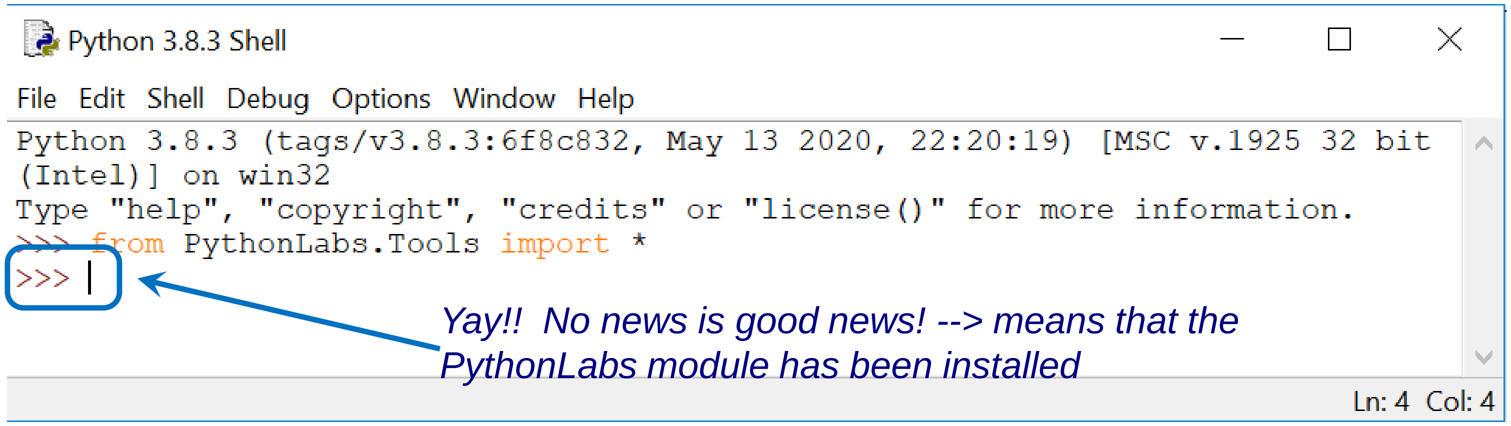


Set up PythonLabs (optional)

- A "module" is a bunch of code written by someone
  - In this case, "PythonLabs" is a module written by JSC and you need this module to if you want to follow all the "tutorial projects" in JSC.
  - Otherwise there is no need to install PythonLabs
- How to install PythonLabs:
  - see http://ix.cs.uoregon.edu/~conery/eic/python/installation.html (go to section on "Install PythonLabs")

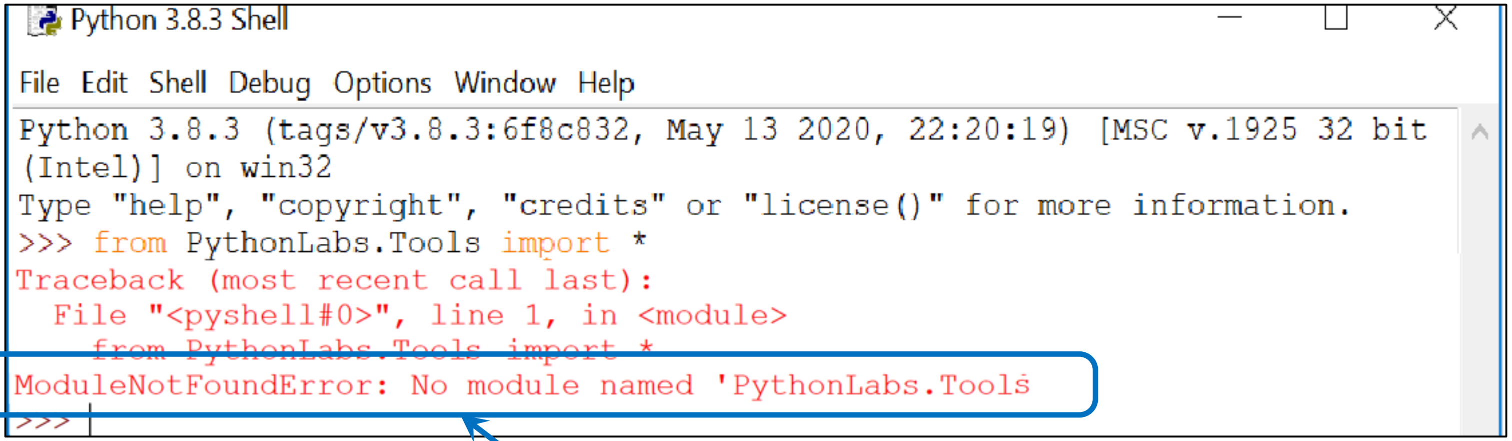
…Have you set up PythonLabs Correctly?

Type this in IDLE (case is important): **from PythonLabs.Tools import ***



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> from PythonLabs.Tools import *
>>> |
```

Yay!! No news is good news! --> means that the PythonLabs module has been installed



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> from PythonLabs.Tools import *
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    from PythonLabs.Tools import *
ModuleNotFoundError: No module named 'PythonLabs.Tools'
>>> |
```

Error message: The PythonLabs module hasn't been installed yet :-(

Set up Other IDE (optional)

- For this course:
  - Demonstrations will be shown using IDLE

    Get familiar with IDLE during this session

    Use only Python 3.x (not Python 2)
  - There are other code editors (or IDE) that you may use for your project:

    NotePad++ (Windows only - search for "run python 3 from notepad++")

    Sublime Text (Win & MacOS)

    Visual Studio Code (Win & MacOS)

    Pyzo (mentioned as "IEP" in JSC) (Win & MacOS)

    Anaconda (?)

Reference

- "Explorations in Computing" by John S. Conery (JSC)
  - Publisher: Taylor & Francis CRC Press
  - Homepage: http://ix.cs.uoregon.edu/~conery/eic/python/index.html
  - PDF: ftp://ftp.cs.uoregon.edu/pub/conery/freeman/EiC.Mar.27.pdf

  - The programming lessons for weeks 1 and 2 follow chapters 2 (Python Workbench) and 3 (Sieve of Eratosthenes) closely.
  - Remaining lessons of this course will not be based on this textbook, but it's still an easy introductory read for some of the topics that will be covered.

Learning Outcomes

- By the end of this session, you should:
  - Be familiar with basic programming concepts in the context of Python (variables, methods, conditionals, while-loops)
  - Be able to <u>**write a simple Python program**</u> to solve a simple problem
  - Be aware of the:

    programming-related requirements for this course (<u>**labs**</u>, <u>**project**</u>)

    programming-related support/resources that we will provide

Computer Programs

- A program is a sequence of instructions written in a particular programming language that the computer runs to complete a task.
- Tools required:
  - Python 3.x
  - PythonLabs (module written by JSC)
  - A code editor (e.g. IDLE)

Python

- Simple, case-sensitive, object-oriented, modern
- You can :
  - Run Python in Interactive mode (e.g. using IDLE), or
  - Run a Python file (<filename>.py) the “normal” way from a terminal window:

    ```
    c:\> python test.py
    ```

- IDLE:
  - Like a calculator
  - Evaluates expressions
  - Four arithmetic operations:

    Addition + , subtraction -, multiplication *, division /

    Use round brackets if necessary in your arithmetic expression

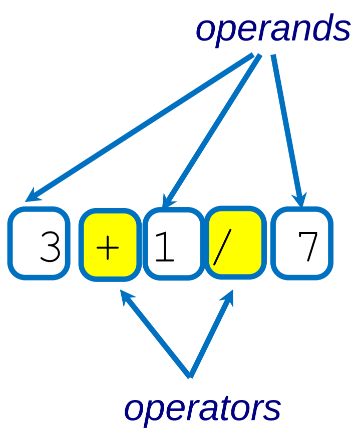
In-class Ex: Let's Get Our Hands Dirty!

- Start IDLE
- Try Tutorial Project T1-8 on **<u>p.36</u>** of the PDF (p.22 of textbook)
  - ftp://ftp.cs.uoregon.edu/pub/conery/freeman/EiC.Mar.27.pdf

Reflecting on Tutorial Project T1-8

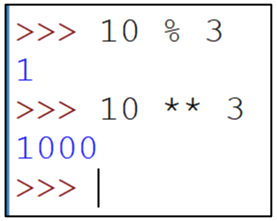
- There seems to be precedence rules:
  - Multiplication and division have higher precedence than addition and subtraction
  - Round brackets are used if necessary
  - Search for "python operator precedence table"

- Terminology: operator, operand



- Other useful operators: ** (exponentiation)

  % (mod, or remainder)



```
>>> 10 % 3
1
>>> 10 ** 3
1000
>>> 
```

…Reflecting on Tutorial Project T1-8

- Terminology:

  Integer = whole number (e.g. 3, -1).

  Float = decimal number (e.g. 3.0, 3.14, -6.9)

- An integer divided by another integer always returns a floating point no.

```
>>> 3/2
1.5
>>> 3/3
1.0
```

- There is a “floor division” operator that returns an integer from a division: ***//***

```
>>> 4/3
1.3333333333333333
>>> 4//3
1
>>> 5/3
1.6666666666666667
>>> 5//3
1
```

Returns 1, not 2 --> truncates, not round

In-class Ex: T9-19

- Try Tutorial Project T9-19 on **p.40** of the PDF (p.26 of textbook)

Reflecting on Tutorial Project T9-19

- You have seen 2 "types" of data: floats and integers.

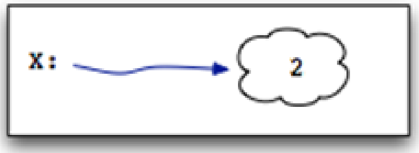
  int * int = int

  float * int = float
- Use **sqrt()** to find the square root of an integer
- **sqrt()** is what we call a "function".
  - Someone has written some code and placed that code in a function
  - You "call" or "invoke" that function in order to run that code
- T15 requires you to "import sqrt", before using the **sqrt()** function. Try to call **sqrt()** without first importing **sqrt()** and see what happens.
  - You need to import the necessary module that a particular function is in before calling that function.

Variables

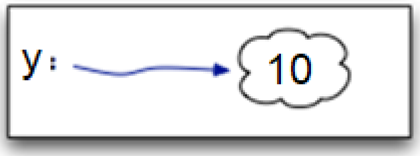
- Variables are temp "storage areas" for a value. You use an assignment statement to store a value into a variable.
- Use the = operator to assign.

```
>>> x = 2
>>> x + 4
6
>>> x + 1
3
>>> x
2
>>> y = 10
>>> x + y
12
>>> y
10
```



Circled in red: assignment statement. x is the name of a variable

These statements do NOT change the value of the variable. They are not assignment statements.



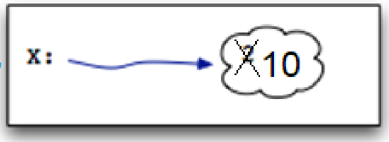
Another assignment statement. y is the name of a 2nd variable

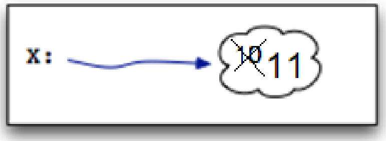
…Variables

- You can change the value of a variable.

  Simply assign it a new value in another assignment statement:

```
>>> x = 10
>>> x
10
>>> x + 1
11
>>> x
10
>>> x = x + 1
>>> x
11
```



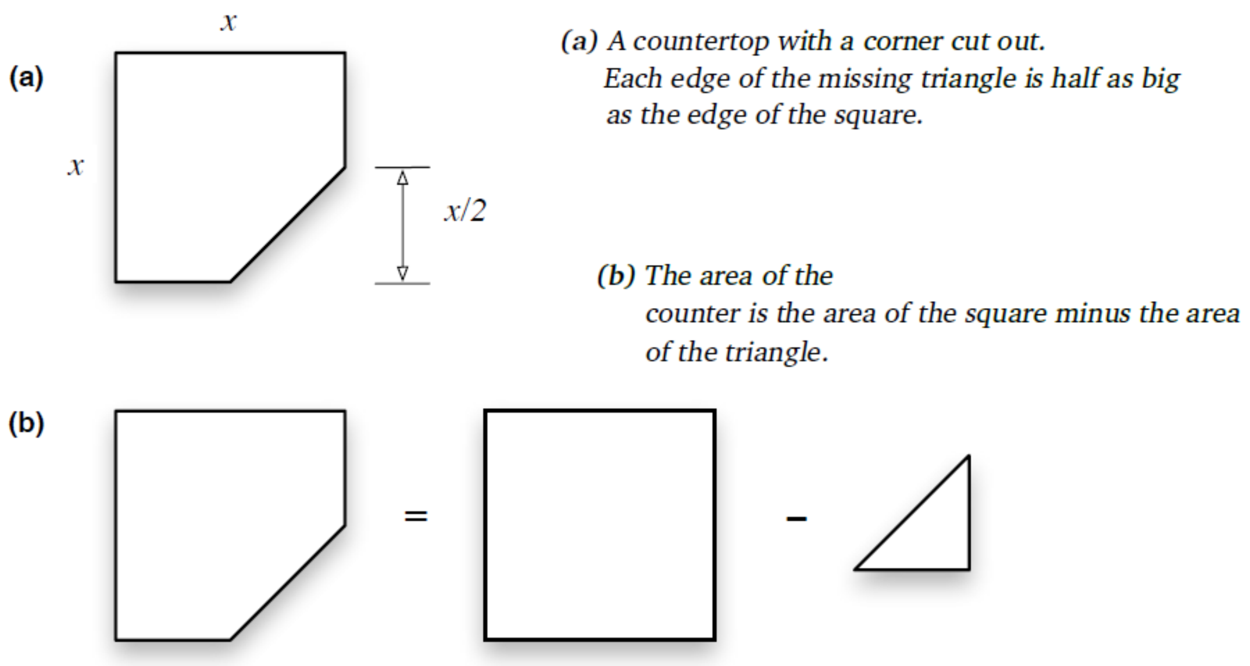


Variable Names

- There are a few rules for defining variable names:
  - Names must start with a letter
  - You can have a mix of upper & lowercase letters, digits, or underscores
  - case is important (**a** is not the same as **A**)
- Naming conventions in Python for variables (and functions):
  - Use all lowercase letters with words separated by underscores to improve readability. e.g. **no_of_students**
  - Some programmers prefer **noOfStudents** (camel notation). This is also ok, as long as you keep to the same notation consistently.

In-class Ex: T20-30

- The next Tutorial Project that you will attempt has got to do with computing the area of the countertop:



(a) A countertop with a corner cut out. Each edge of the missing triangle is half as big as the edge of the square.

(b) The area of the counter is the area of the square minus the area of the triangle.

- Try Tutorial Project T20-30 on **p.44** of the PDF (p.30 of textbook)

Reflecting on Tutorial Project T20-30

- Remember that assignment statements are not mathematical formulas:
  - T24:

```
>>> x = 109
```

  - T27:

```
>>> triangle = ((x/2) ** 2) / 2
>>> triangle
10395.875
```

  - T29:

```
>>> x = 107
```

Does this change the value stored in triangle?

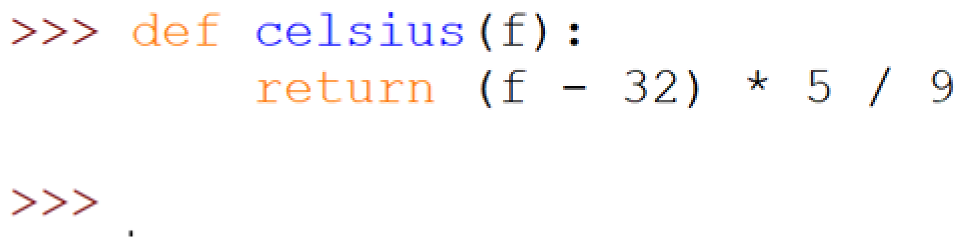
Summary so far…

- What we have just covered:
  - Variables
  - Types (strings, integers, floating point numbers, booleans)
  - Arithmetic operations

COR-IS1702: Computational Thinking
Week 2a: Python Workbench

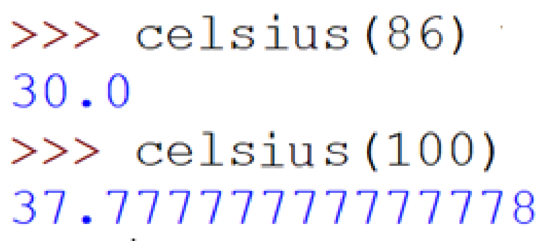
Writing Your Own Functions

- You have used the **sqrt()** function. Someone wrote it, and you called it.
- Now you are going to write (or define) your own function for others (or yourself) to call.
- You can define a function in IDLE:



```
>>> def celsius(f):
        return (f - 32) * 5 / 9

>>>
```

- And then use it:



```
>>> celsius(86)
30.0
>>> celsius(100)
37.77777777777778
```

…Writing Your Own Functions

- But that's a bad idea because you cannot edit the function.
- So, you will want to write your function in a Python file (called **<something>.py**) and save it somewhere.
- IDLE:
  - From menu bar: File --> New File
  - Type your function definition in the new window that pops up
  - Save it as **test.py** (or <any_other_name>.py) in a working folder (e.g. c:\temp or c:\is103).
- To run the code in the file:
  - From the menu bar of **test.py**: Run --> Run Module (or hit F5)
  - This will "place" the **celsuis()** function in memory
- Once your restart the IDLE Shell (Shell --> Restart Shell), all the variables and functions in memory will be gone.

In-class Ex: T31-40

- The next Tutorial Project that you will attempt requires you to write 2 functions:

  celsius()

  countertop()

- Try Tutorial Project T31-40 on **p.49** of the PDF (p.35 of textbook)

Reflecting on Tutorial Project T31-40

Arguments (values passed into a function)

function definition

```
def f1(x):
    return x * 2
```

when calling it:

```
f1(10)
```

A function can take in multiple values:

function definition

```
def f2(x, y):
    return x * y
```

x corresponds to 10,
y corresponds to 20

when calling it:

```
f2(10, 20)
```

…Reflecting on Tutorial Project T31-40

Alternatively, when calling a function, you may specifically state which argument(value) is to be assigned to which parameter. This works as well:

function definition

```
def f1(x):
    return x * 2
```

when calling it:

```
f1(x=10)
```

---

function definition

```
def f2(x, y):
    return x * y
```

when calling it:

```
f2(x=10, y=20)
```

…Reflecting on Tutorial Project T31-40

- You can have multiple statements in a function. The statements in the function will run sequentially. E.g.

```
def f3(x, y):
    z = x * y
    return z
```

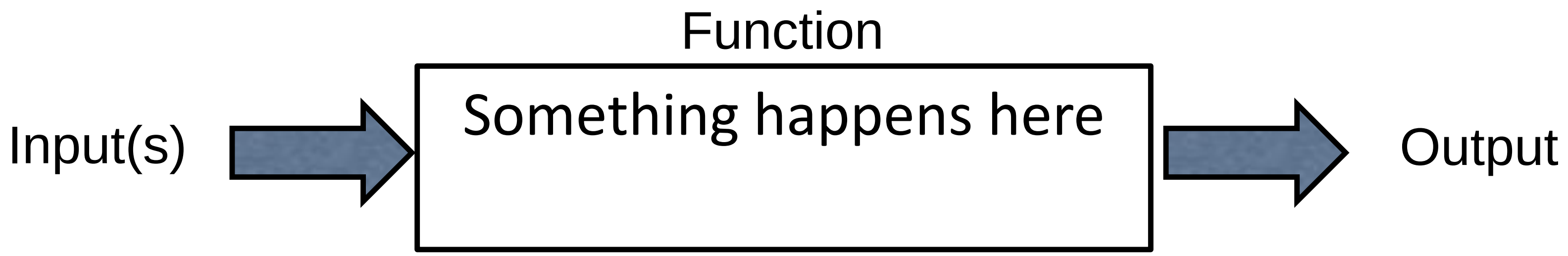
- **return** and **def** are "reserved words" or "keywords" in Python. You cannot choose keywords as variable or function names.
- In Python, the start and end of functions are not demarcated by symbols. The way you indent the statements will tell Python where the function starts and ends.
- You can have a function that takes in nothing
- You can have a function that doesn't have a return statement (and hence return **None**).

…Reflecting on Tutorial Project T31-40

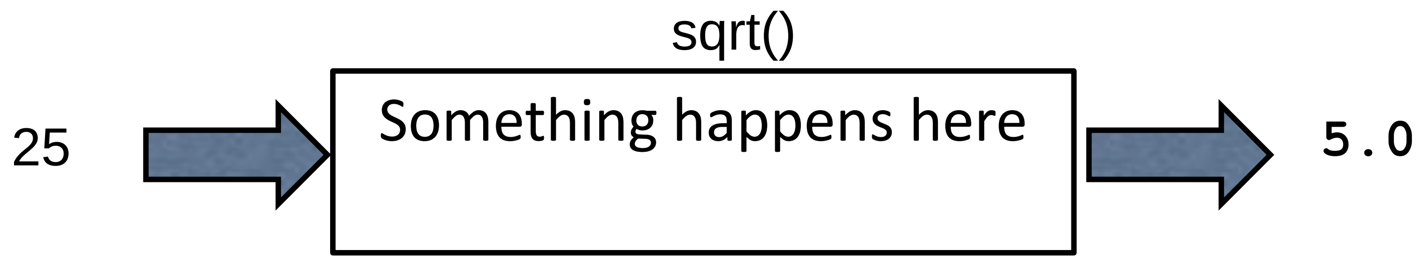
- You can insert "docstrings" in a function. Docstrings are helpful hints to programmers who use your function as to what it does.
  - Use help(<function_name>) to see the docstring for that function.
  - Try: **help(sqrt)**

...Reflecting on Tutorial Project T31-40

A function is like a "system":



A system takes in something, does something, and may return something.



Call the sqrt() function and assign the returned value to a variable called answer:

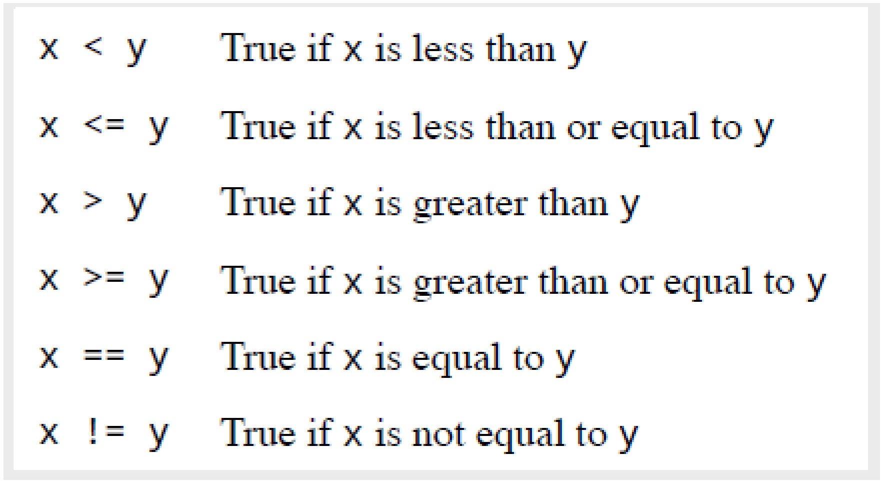
```
answer = sqrt(25)
```

Boolean Expressions

- A Boolean expression is a combination of values, variable names and operators, and
  - Always evaluates to either **True** or **False**.
- E.g. of Boolean expressions:
  - 10 > 5
  - 10 < 10
  - 10 <= 10
  - x == 10
- These are Boolean operators:

  <, <=, >, >=, ==, !=
- Important:

  == is not the same as =



x < y	True if x is less than y
x <= y	True if x is less than or equal to y
x > y	True if x is greater than y
x >= y	True if x is greater than or equal to y
x == y	True if x is equal to y
x != y	True if x is not equal to y

Conditional Execution

- New keywords: **<u>if</u>**, **<u>else</u>**
- **tax_rate** is a function to compute tax rate depending on income level:
  - People who earn <u>less than</u> $10k will be taxed at 0%, but
  - People who earn $10k or more will be taxed at 5%

```
def tax_rate(income):
    if income < 10000:
        return 0.0
    else:
        return 5.0
```

…Conditional Execution

- New keyword: **elif** (else if)
- Let's say the tax rate calculation rules have changed to 3 tiers:
  - People who earn less than $10k will be taxed at 0% (same), but
  - People who earn $10k to below 20k will be taxed at 5%, and
  - People who earn $20k or more will be taxed at 7%.

```
def marginal_tax_rate(income):
    if income < 10000:
        return 0.0
    elif income < 20000:
        return 5.0
    else:
        return 7.0
```

In-class Ex: T41-46

- Try Tutorial Project T31-40 on **p.52** of the PDF (p.38 of textbook)

Reflecting on Tutorial Project T41-46

- An **if** block may contain multiple statements. E.g. this is OK:

```
def f1(x):
    if x > 10:
        print("bigger than 10")
        return True
    else:
        print("smaller or equal 10")
        return False
```

- When calling a function, once the program reaches a "return statement", the function terminates immediately.

…Reflecting on Tutorial Project T41-46

- This function doesn't make sense because lines 4, 5 and 8 will never get a chance to run. But will not cause an error:

```
1 def f(x):
2   if x>10:
3     return 1
4     return 4
5     xxxxxxx
6   else:
7     return 2
8   return 3
```

…Reflecting on Tutorial Project T41-46

- It is possible to use **if** without **else** or **elif**:

```
def f(x):
  if x>10:
    return 1
  return 3
```

…Reflecting on Tutorial Project T41-46

- It is possible to use if without else or elif:

```
1 def f(x):
2     if x>10:
3         return 1
4     return 3
```

To be viewed as 1 statement.

When line 2 is executed, and the Boolean expression is False (i.e. x>10 is False), the program will move to line 4.

Strings

- We have seen integers, floats and booleans. Data can be stored as strings as well.

  A string is a set of characters (letters, digits, punctuation marks, spaces etc.) enclosed in quotes.

- We can store strings in variables:

```
>>> s = "hello"
```

- We can do interesting things with strings:

```
>>> "pine" + "apple"
>>> s + "world"
>>> s * 3
>>> "hell" in s
>>> len(s)
>>> str.count(s,"l")
>>> s.count("l")
```

String concatenation

Returns a Boolean depending on whether "hell" is a substring of s

Returns the no. of characters in s

In-class Ex: T47-65

- Try Tutorial Project T47-65 on <u>**p.55**</u> of the PDF (p.41 of textbook)

- Typos:
  - T48:

    Code given for plural.py line 1: should be "w" instead of "word"
  - T49:

    ```
    len((s + '! ') * 2)
    ```

    Exclamation mark and a space

Reflecting on Tutorial Project T47-65

- Anything to share?

- How do you include a quotation mark in a string?
  - Search for "python escape characters"

In-class Ex: T66-71

- Try Tutorial Project T66-71 on **p.60** of the PDF (p.46 of textbook)

  - Typo. T68 should be:

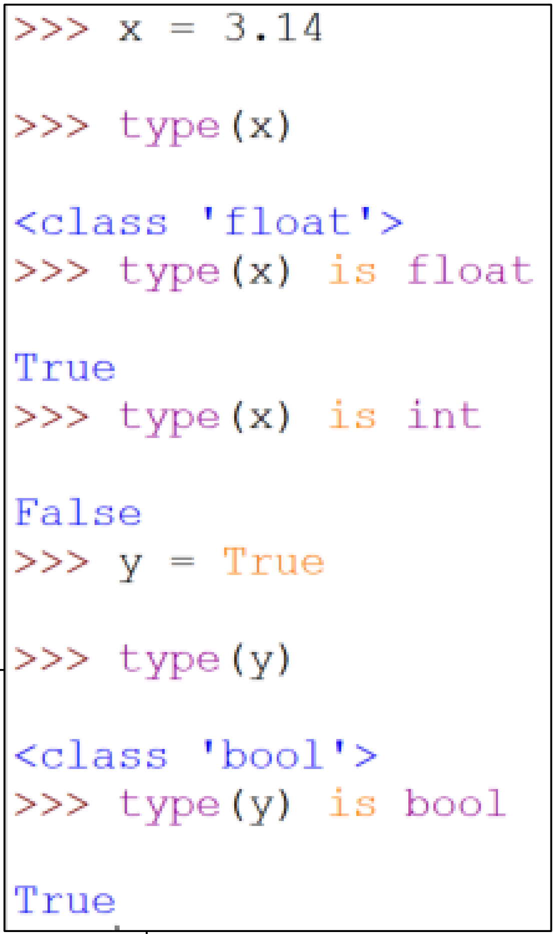
    ```
    x ** 2
    ```

    Instead of

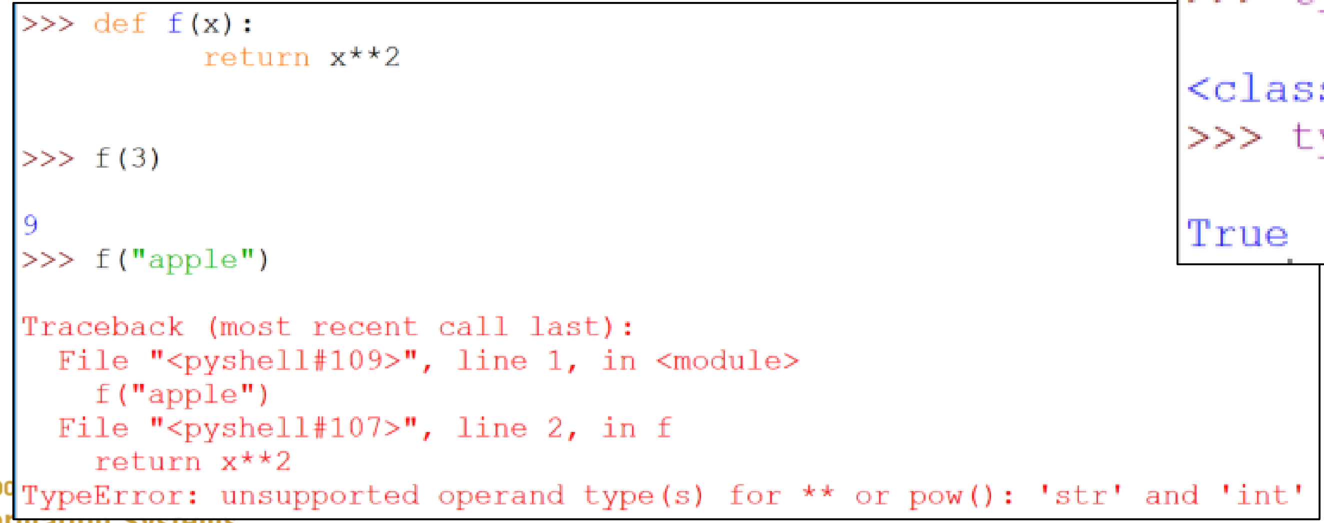
    ```
    s ** 2
    ```

Reflecting on Tutorial Project T66-71

- In Python, a variable can be used to store a value of any type:
  - A variable used to store an integer may be reassigned to store a string
  - You can check if a variable is of a particular type by using the **is** keyword -->
  - In order to write a more robust function, you may want to check if the value that the caller has passed in is of a certain type.



```
>>> x = 3.14

>>> type(x)

<class 'float'>
>>> type(x) is float

True
>>> type(x) is int

False
>>> y = True

>>> type(y)

<class 'bool'>
>>> type(y) is bool

True
```



```
>>> def f(x):
        return x**2


>>> f(3)

9
>>> f("apple")

Traceback (most recent call last):
  File "<pyshell#109>", line 1, in <module>
    f("apple")
  File "<pyshell#107>", line 2, in f
    return x**2
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

Summary so far…

- Last week:
  - Variables
  - Types (strings, integers, floating point numbers, booleans)
  - Arithmetic operations
- Just Now:
  - Functions:

    calling a function and

    defining a function
  - Conditionals (if, elif, else)