DATA TYPES, ERRORS AND DEBUGGING, ADVANCED MATH OPERATIONS & FORMATTING OUTPUT
Data Types
Data Types

- Python needs to know how to set aside memory in your computer based on what kind of information you want to store.

- There are three basic types of data that we will be working with during the first half of the term:
  - Strings (character-based data)
  - Numbers
  - Logical Values (True / False)
Numeric Data Types

- **Integers**
  - Whole numbers that do not contain a decimal point
  - Abbreviated as “int” in Python
  - Example: 5, -5, 100, 10032
  - Note: often referred to with a ‘d’ in Python for decimal integer

- **Floating Point Numbers**
  - Numbers that contain a decimal point
  - Abbreviated as “float” in Python
  - Example: 5.0, -5.0, 100.99, 0.232132234
Numeric Data Types

- You can store numeric data inside variables that you create. Example:

  ```
  num_1 = 5  # this is an int
  num_2 = 4.99  # this is a float
  ```

- Keep in mind that you do not use separators or symbols when storing numeric data. Example:

  ```
  num_3 = $5,123.99  # error!
  ```
What's the data type?

5
5.5
"Hello"
"5.5"
2.975
2.0
True
Numeric Data Types

- Python is not a strictly typed language. This means that you don’t need to pre-declare what kind of data your variables will be holding.
- This is also called “dynamic typing”.
# Data Types across languages

<table>
<thead>
<tr>
<th>Loosely Typed</th>
<th>Strictly Typed</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ Python</td>
<td>✔ C</td>
</tr>
<tr>
<td>✔ PHP</td>
<td>✔ C++</td>
</tr>
<tr>
<td>✔ JavaScript</td>
<td>✔ Java</td>
</tr>
<tr>
<td>✔ Perl</td>
<td></td>
</tr>
</tbody>
</table>
Strictly Typed Languages - Examples

**ActionScript**

```actionscript
var name:String = "Harry";
var top_speed:Number = 50;
var gravity:Number = 9.5;
```

**Java**

```java
String name = "Harry";
int top_speed = 50;
float gravity = 9.5;
```
We can capture input from the user (via the `input()` function) and use that input in our calculations.

However, the `input()` function “returns” a string – this means that the data type that “comes out” of the `input()` function is a series of printed characters.

We need to convert the result of the `input` function from a string into one of the two numeric data types that Python supports (float and int).
Solution: The `float()` and `int()` functions

- `float()` and `int()` are data type conversion functions. They each take one argument and convert that argument into the specified data type.

- Example:

```python
# ask the user for their monthly salary
monthly_salary = input('how much do you make in a month?')

# convert the salary into a float
monthly_salary_float = float(monthly_salary)

# calculate the yearly salary
yearly_salary = monthly_salary_float * 12

# print the result
print ('that means you make', yearly_salary, 'in a year')
```
Nesting data type conversions

- In the previous example we performed our data type conversion in two lines
- We could have done that in a single line using a technique called “nesting”
- Example:

```python
mynum = float(input('give me a number!'))
```
Nesting data type conversions

```python
my_num = float(input('give me a number!'))
```
I’m asking people for monthly salary. Sometimes they give me with cents, but I want to round it to nearest dollar. How?
Errors, Bugs and Debugging
“It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and [it is] then that 'Bugs' — as such little faults and difficulties are called—show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached.”

- Thomas Edison, 1878
De-bugging a program is the process of finding and resolving errors.
Types of Errors

- **Syntax errors**: The code does not follow the rules of the language; for example, a single quote is used where a double quote is needed; a colon is missing; a keyword is used as a variable name.

- **Runtime errors**: In this case, your code is fine but the program does not run as expected (it “crashes”). For example, if your program is meant to divide two numbers, but does not test for a zero divisor, a run-time error would occur when the program attempts to divide by zero.

- **Logic errors**: These can be the hardest to find. In this case, the program is correct from a syntax perspective; and it runs; but the result is unanticipated or outright wrong. For example, if your program prints “2+2 = 5” the answer is clearly wrong 😊
Example Errors

print ("hello, world!")
Example Errors

```python
print ("hello, world!")
```

Syntax error (delimiters don’t match)
num = input('give me a number: ')
num_float = float(num)
new_num = 10 + num_float
print(new_num)

give me a number: apple
Traceback (most recent call last):
File "/Users/HarryPotter/Documents/madlibs01.py", line 6, in <module>
    new_num = 10 + num
TypeError: unsupported operand type(s) for +: 'int' and 'str'
Example Errors

Source

```python
num_1 = float(input('give me a num: '))
num_2 = float(input('give me another num: '))
print('the sum is: ', num_1 - num_2)
```

Execution

give me a num: 5
give me another num: 2
the sum is: 3.0
Basic Debugging Techniques

- Set small, incremental goals for your program. Don’t try and write large programs all at once.
- Stop and test your work often as you go. Celebrate small successes
- Use comments to have Python ignore certain lines that are giving you trouble
Advanced Math Operations
Division Operations

- Python contains two different division operators
- The “/” operator is used to calculate the floating-point result of a division operation
- The “//” operator is used to calculate the integer result of a division operation (essentially throwing away the remainder). This operation will always round down.
- Most times you will use the floating point division operator (“/”)
Division Operations

print (5/2)  # 2.5
print (5//2)  # 2
print (-5/2)  # -2.5
print (-5//2)  # -3
Order of Operations

- Python supports the standard order of operations (PEMDAS)
- You can use parenthetical notation inside your math expressions to group operations
- Ex:

  \[
  \frac{(5+10+20)}{60} \times 100
  \]
Exponents

- You can raise any number to a power by using the “**” operator.
- Example: $2^4$

2 ** 4
Remainder Operator (modulo)

- The modulo operator ("%") returns the remainder portion of a division operation.

- Example:

  \[ 5 / 2 \quad \# \quad 2.5 \]

  \[ 5 \% 2 \quad \# \quad 1 \]
Math formulas need to be converted into a format that Python can understand before they can be evaluated.

<table>
<thead>
<tr>
<th>Math Expression</th>
<th>Converted to Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10b$</td>
<td>$10 * b$</td>
</tr>
<tr>
<td>$(3)(12)$</td>
<td>$3 * 12$</td>
</tr>
<tr>
<td>$4xy$</td>
<td>$4 * x * y$</td>
</tr>
<tr>
<td>$y = 3 \frac{x}{2}$</td>
<td>$y = 3 * x / 2$</td>
</tr>
</tbody>
</table>
Mixed Type Expressions

- Python allows you to mix ints and floats when performing calculations.
- The result of a mixed-type expression will evaluate based on the operands used in the expression.
# Mixed Type Expressions

<table>
<thead>
<tr>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>float</td>
<td>int</td>
<td>float</td>
</tr>
</tbody>
</table>
Sometimes expressions can get to be very long

- You can use the “\” symbol to indicate to Python that you’d like to continue the expression onto another line

- Example:

```
x = 5 + 2 / 7 \n    + 8 - 12
```

- This also works for print() function calls as well
Escape Characters

- The ‘\’ is commonly referred to, in many programming languages, as the signifier escape character. It says ‘after this, this is not meant to be written in the string’.

- Escape characters include:
  - \n → new line
  - \t → new tab position
  - ‘ → Print a single quote that’s not meant to end a string
  - \" → Print a backslash that isn’t supposed to be an escape character
String Manipulation
String manipulation

- We are going to really get into how to manipulate strings as the semester goes on.
- Python is a particularly good language to deal with strings. Possible examples include:
  - Sort alphabetically
  - Find strings that start with ‘Super duper’
  - Count the amount of times an excerpt of text uses the word ‘the’
  - Parse out the ‘stop words’ in an excerpt of text
String Concatenation

- You can’t subtract ‘apple’ from ‘pear’ because it doesn’t make sense
- You can’t “add” strings together either, but you can “concatenate” them into a single compound string
- Example:

```
a = input('first name')
b = input('last name')
c = b + ',' + a
print (c)
```
You can also “multiply” a string by an integer value to produce a larger string.

Example:

```python
lyrics = 'Fa ' + 'La ' * 8
print (lyrics)
```

# Fa La La La La La La La

Notice the intentional spaces!
The `format()` function
Formatting Strings

- The `format()` function can be used to format a string before you decide to print it out to the user.
- `format()` takes two arguments – a string literal or variable and a formatting pattern (expressed as a string).
- `format()` returns a string which can be treated like any other string (i.e. you can print it out immediately, store its value in a variable, etc).
One common use of string formatting is to generate a string that contains a known # of characters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Occupation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batman</td>
<td>Caped Crusader</td>
</tr>
</tbody>
</table>

In this case we need to ensure that the strings “Name” and “Batman” are the same width so that the strings that come after them (“Occupation” and “Superhero”) line up correctly.

```python
name = format(‘Batman’, ‘<6s’)
```
String Formatting

- Full example:
  ```python
  heading1 = format('Name', '<6s')
  heading2 = format('Occupation', '<16s')
  name = format('Batman', '<6s')
  job = format('Caped Crusader', '<16s')
  ```

- This will generate new strings that contains the necessary spaces. The total length of a line in this case will be 22 characters (plus a space if using one print function per line).

- The '<' character in the formatting pattern tells Python to left justify the string and place the extra spaces at the end of the new string. '>' would be to the right.
The format() function can also be used to generate a printable string version of a float or integer number.

- format() takes two arguments – a number and a formatting pattern (expressed as a string).

- format() returns a string which can be treated like any other string (i.e. you can print it out immediately, store its value in a variable, etc).
Limiting decimal precision

```python
>>> a = 1/6
>>> print(a)
0.16666666666666666

>>> # Format to 2 decimal places
>>> b = format(a, '.2f')
>>> print(b)
0.17
```
a = 10000/6
# format a as a 2 digit float
b = format(a, '.2f')

# format a as a 5 digit float
c = format(a, '.5f')

# 5 digit float + comma separators
d = format(a, ',.5f')

'''2 digit float, commas + 20 character minimum field width, justified to the right'''
e = format(a, '>20,.2f')
>>> a = 0.52

>>> print(format(a, '%'))
52.000000%

>>> print(format(a, '.2%'))
52.00%

>>> print(format(a, '.0%'))
52%
>>> a = 20000

>>> print(format(a, ',d'))
20,000

>>> print(format(a, '>20,d'))
20,000
Turtle
A lot of new programming languages are combining art and technology and make creativity in coding easier

Languages such as Processing, p5, Scratch, as well as hobbyist tools such as Arduino, Little Bits, and Raspberry Pi are very successfully doing this

Python doesn’t lend itself as easily (though Raspberry Pi’s are usually coded in Python), but it does have many modules to allow for creative expression
Drawing with code

- Python uses the Cartesian coordinates system to draw – a bit more mathematic than the commonly used pixel coordinate system.
Python only has a certain amount of key words, to give our programs more ability we can import modules with new keywords that can add more functionality.

For drawing with turtle, we need to import it at the top of our script:

```python
import turtle
```
Useful turtle commands

# setting up a turtle canvas
turtle.setup(width, height)

# orientation functions
turtle.right(degrees)
turtle.left(degrees)
turtle.setheading(degrees)

# movement functions
turtle.forward(pixels)
turtle.goto(x_position, y_position)

# drawing control
turtle.penup()
turtle.pendown()

# set up our pen
turtle.pencolor("red")
turtle.pensize(5)

# set up our fill color
turtle.fillcolor("yellow")

# tell Python to start filling
turtle.begin_fill()

# tell Python that are we are finished filling
turtle.end_fill()
Ask the user for two numbers. You can assume they will be floating point numbers.

Compute the following and print it out to the user:
- The sum of the numbers
- The product of the numbers
- The difference between the numbers
- The first number divided by the second number
Programming Challenge - Coins

- Write a program that asks the user for a number of pennies, nickels, dimes and quarters
- Calculate the total amount of money that the user has and print it out
Programming Challenge – Subway Ride Calculator

- Write a program that asks the user for the value of their current Metro card
- Compute how many rides they have left on their card. Only provide whole number results (i.e. you cannot have 3.5 rides left on a card)
Programming Challenge

- Write a program that asks the user for three price values.
- Calculate the average price in a single variable and output it to the user.
Programming Challenge: Calculating the area of a square

Area $= s^2$

**(Area = length x width. For a square the length and width are the same.**)

Example:

6 in
Programming Challenge: Time Calculations

Ask the user to input a number of seconds as a whole number. Then express the time value inputted as a combination of minutes and seconds.

Enter seconds: 110
That’s 1 minute and 50 seconds
Programming Challenge: Investment Planning

- In this exercise you will ask the user to input the following values:
  - How much money they want to generate
  - An interest rate value
  - How long they'd like to invest their money

- Calculate how much they will need as an initial investment.

Example:
- You will need ________ dollars to generate ________ dollars at ________ % over ________ years.
Programming Challenge: Investment Planning

- P = Present Value
- F = Future Value
- R = Rate or Return
- N = Number of Years

\[ P = \frac{F}{(1 + r)^n} \]
Programming Challenge

- Write a program that asks the user to enter in 3 products and 3 prices.
- Format your output to look like the following:

```
Product          Price
product1        price1
product2        price2
product3        price3
```
Write a program that generates the 2 times table, like this:

<table>
<thead>
<tr>
<th>Number 1</th>
<th>Number 2</th>
<th>N1 * N2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>