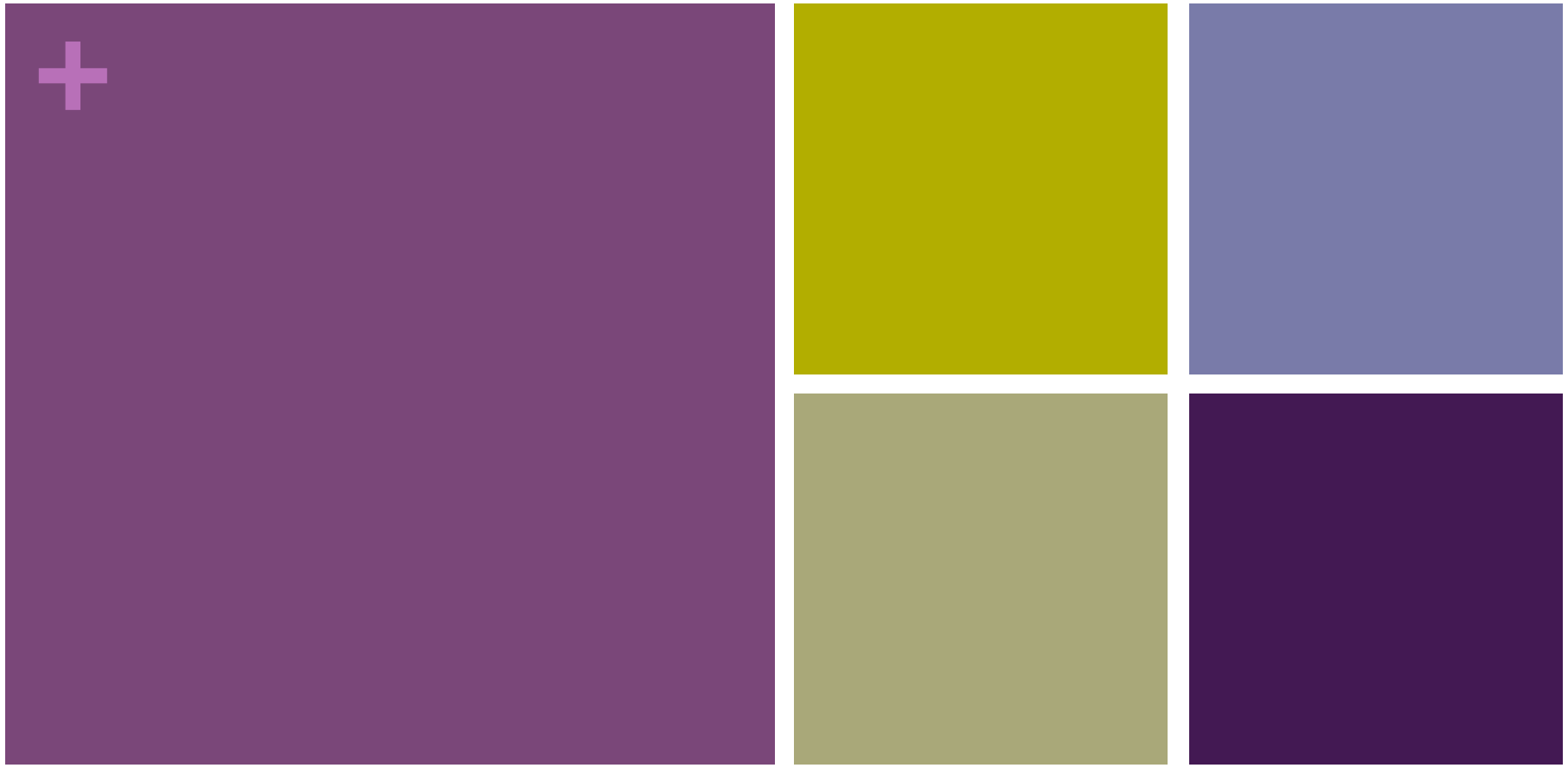


Data Types, Errors and Debugging, Advanced Math Operations & Formatting Output

CSCI-UA.002

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
- 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
```

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

Loosely Typed	Strictly Typed
Python	C
PHP	C++
JavaScript	Java
Perl	ActionScript

Strictly Typed Languages - Examples

ActionScript

```
var name:String = “Harry”;

var top_speed:Number = 50;

var gravity:Number = 9.5;
```

Java

```
String name = “Harry”;

int top_speed = 50;

float gravity = 9.5;
```

User input and Math Expressions

- 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 conversation functions. They each take one argument and convert that argument into the specified data type
- Example:

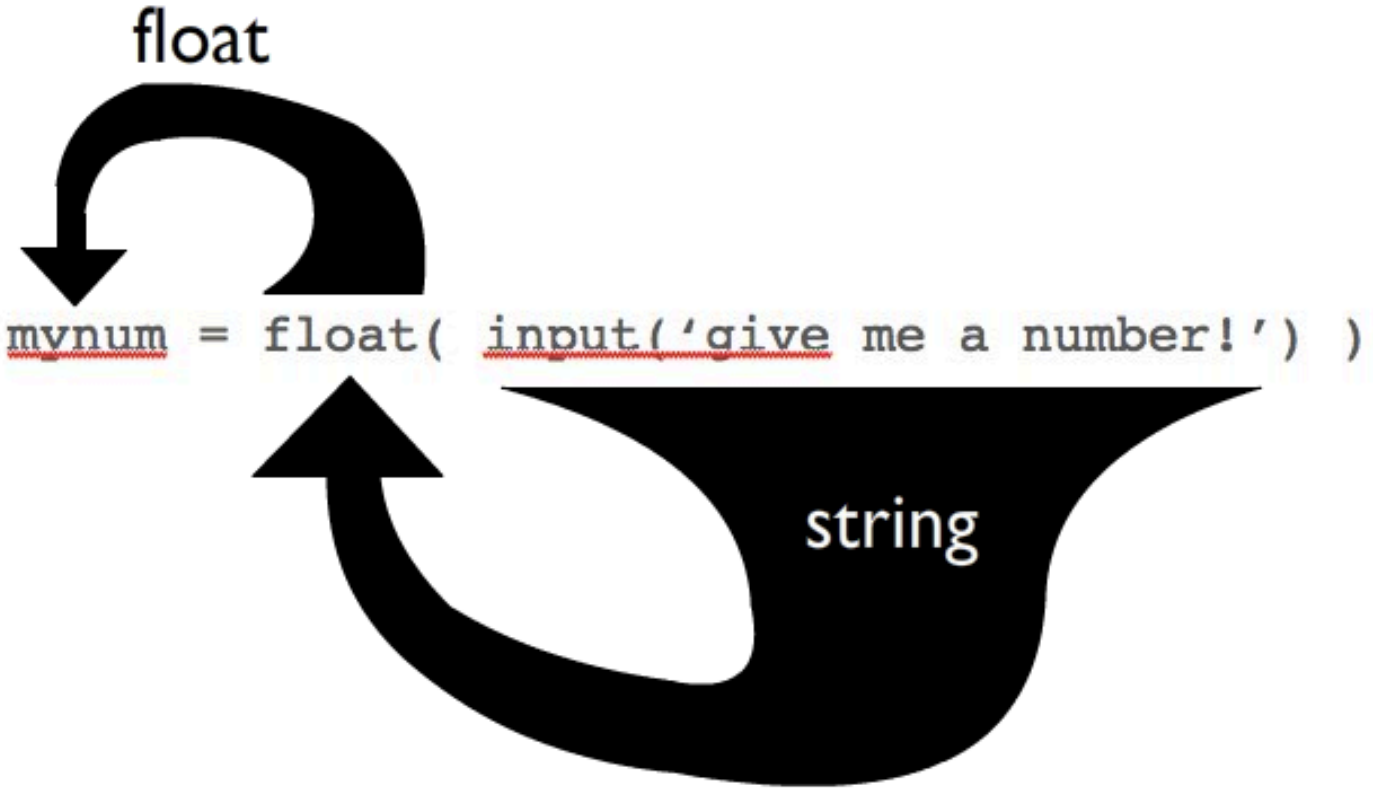
```
# ask the user for their monthly salary
monthly_salary = input('how much do you make in a
month?')

# 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:

```
mynum = float( input('give me a number!') )
```

Nesting data type conversions



Programming Challenge

- 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)

Errors, Bugs and Debugging

The Software Error

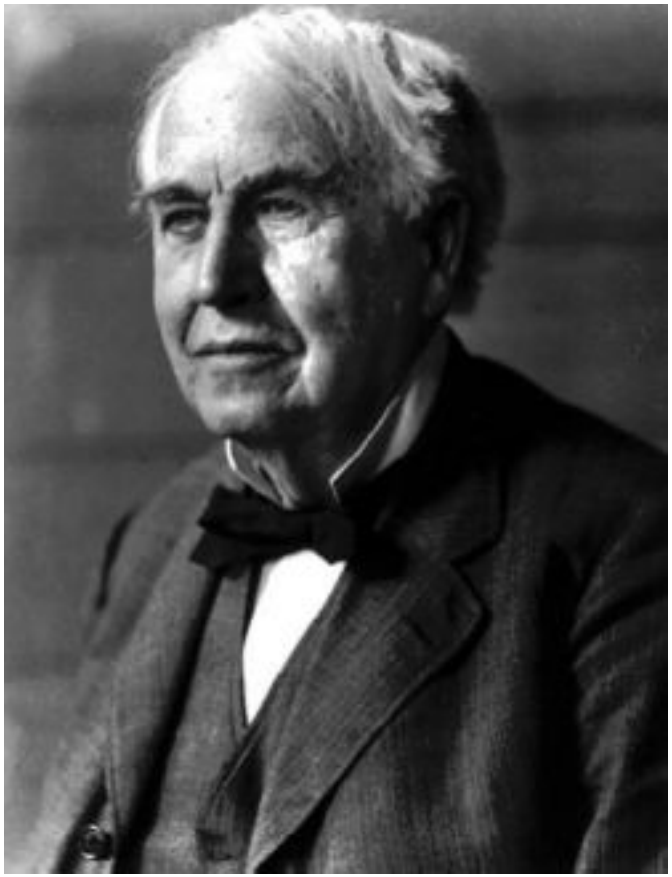
"...an analyzing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of error. **Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders**."

- Lady Augusta Ada King, Countess of Lovelace (1843)

Mechanical Malfunctions

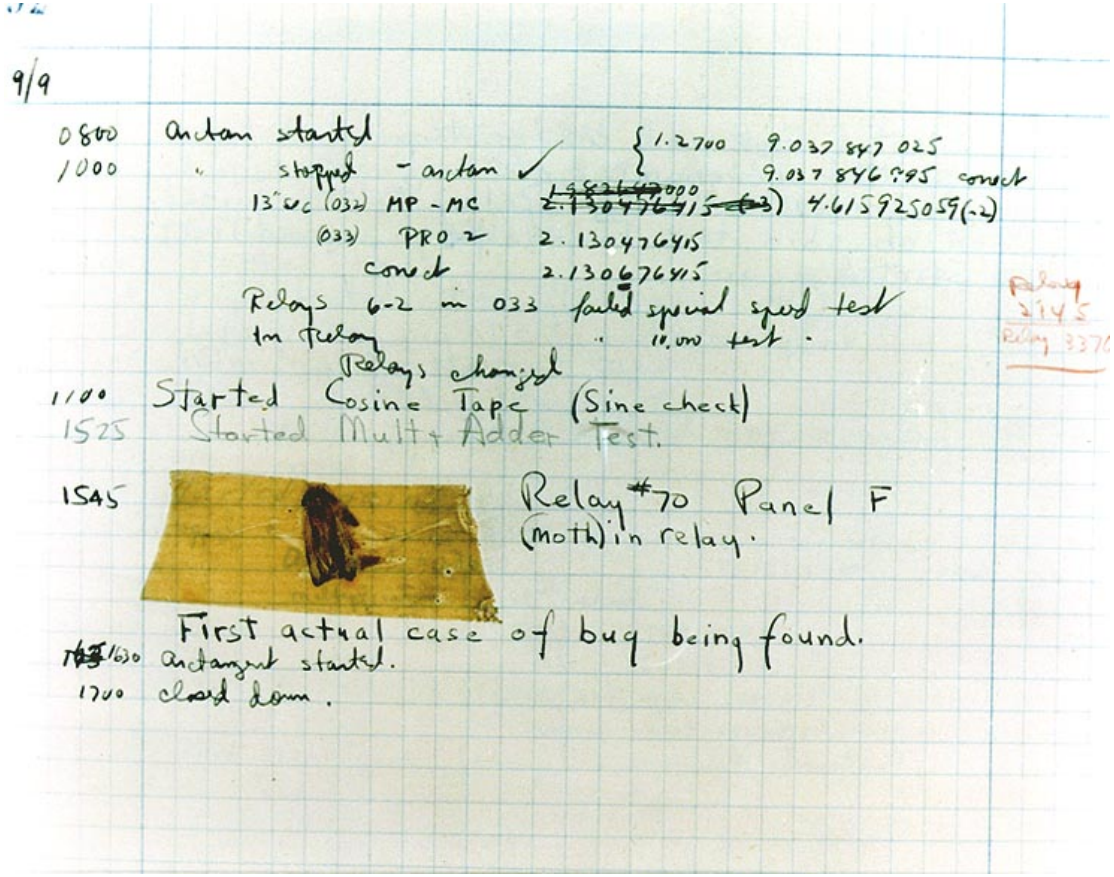
"It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and [it is] then that '**Bugs**' — as such little faults and difficulties are called—show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached."

- Thomas Edison, 1878



"Debugging"

1947, Harvard Mark II Computer



9/9

0800 Arctan started
1000 " stopped - arctan ✓
13°v.c (032) MP - MC
(033) PRO 2
correct
{ 1.2700 9.037 847 025
9.037 846 995 correct
4.6159250597(-2)
2.13047615
2.13067615

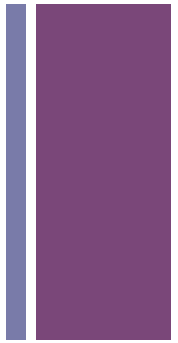
Relays 6-2 in 033 failed special speed test
In relay " 10.00 test.
Relays changed

1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

1700 closed down.

“Debugging”



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

```
print (“hello, world!’)
```

Syntax error (delimiters don’t match)

Example Errors

Source

```
num = input ('give me a
  number: ')

num_float = float(num)

new_num = 10 + num_float

print (new_num)
```

Execution

```
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'
```

(that was a runtime error)

The program ran, but when given bad data it crashed

Example Errors

Source

```
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
```

(that was a logic error)

The program ran, but it didn't do what it set out to do (i.e. it gave us the wrong answer)

Basic Debugging Techniques

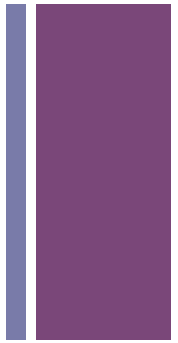
- Set small, incremental goals for your program. Don't try and write large programs all at once.
- Stop and test your work often as you go. Celebrate small successes ☺
- Use comments to have Python ignore certain lines that are giving you trouble

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

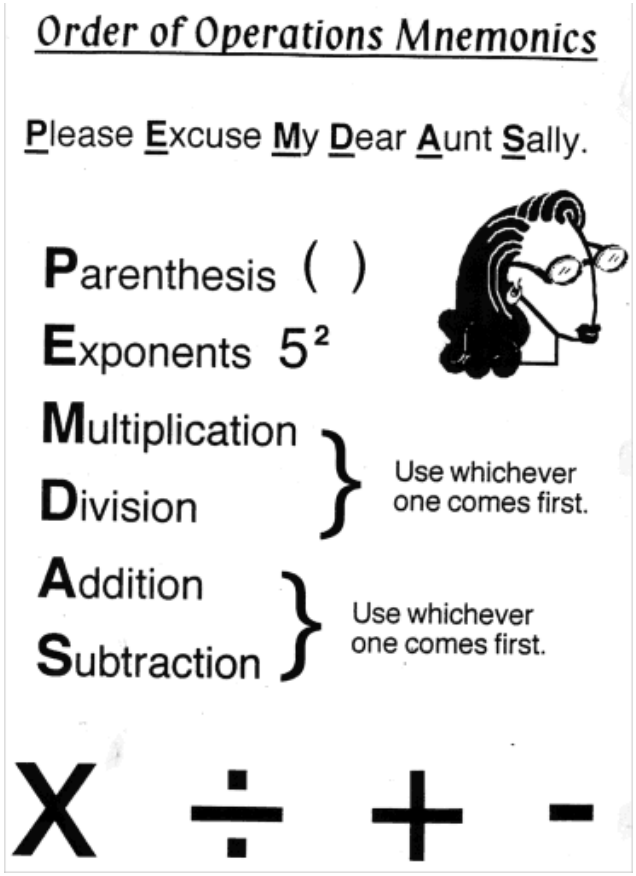


```
print (5/2)                    # 2.5

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:

```
((5+10+20)/60) * 100
```



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

Calculating an average in a single step

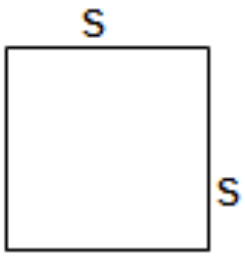
average_score = (100 + 50 + 88) / 300

Exponents

- You can raise any number to a power by using the "**" operator
- Example: 2^4

```
2 ** 4
```

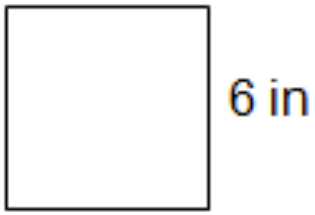
Programming Challenge: Calculating the area of a square



Area = s^2

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

Example:



Remainder Operator (modulo)

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

```
5/2                 # 2.5
5%2                 # 1
```

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
```

Converting Math Formulas into Programming Statements

- Most math formulas need to be converted into a format that Python can understand before they can be evaluated

Converting Math Formulas into Programming Statements

`10b`	`10 * b`
(3)(12)	`3 * 12`
`4xy`	`4 * x * y`
$y = 3\frac{x}{2}$	`y = 3 * x / 2`

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

$$P = \frac{F}{(1+r)^n}$$

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

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

Operand 1	Operand 2	Result
int	int	int
float	float	float
float	int	float

Line Continuation

- 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     \
    + 8 – 12
```

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

Formatting Output with the print() function

Line endings

- When using the print() function you probably have noticed that Python automatically places a newline character at the end of each line
- You can override this behavior and have Python use a character of your choice by using the optional 'end' argument when using the print() function
- Example:

```
print ('one', end='')
print ('two', end='')
```

Separating print() function arguments

- By default, Python will place a space between arguments that you use in print() function calls
- You can override this behavior by using the optional 'sep' argument
- Example:

```
print ('one', 'two', sep='*')

# output:  one*two
```

Combing both line endings and separators

- You can use both the ‘sep’ and the ‘end’ arguments at the same time in the same print() function call.
- Example:

```
print (‘a’, ‘b’, ‘c’, sep=‘*’, end=‘’)
```

Escape Characters

- Most programming languages support an "escape character" that allows you to perform special actions inside the confines of a delimiter.
- In Python the escape character is the "\" character
- It causes Python to treat the next character as a "special" character – in most cases this means that you will ignore the next character and prevent it from interfering with your delimiter
- Example:

```
print ('Hi, I\'m Harry Potter, your professor')
```

Escape Characters

- There are a number of special characters you can use in conjunction with the escape character to perform special string operations.
- Example – "\n" – forces a line break.

```
print ('line 1\nline 2\nline 3\n')

# line 1
# line 2
# line 3
```

Escape Characters

- Example – "\t" – forces a tab:

```
x = 5
y = 4

print ('X', '\t', 'Y', '\t', 'X*Y')
print (x, '\t', y, '\t', x*y)
```

```
X     Y     X*Y
5     4     20
```

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
```

String Concatenation

- You can't "add" strings together, but you can "concatenate" them into a single compound string
- Example:

```
a = input('first name')
b = input('last name')

c = b + ',' + a

print (c)
```

String Repetition

- You can also "multiply" a string by an integer value to produce a larger string
- Example:

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

# Fa La La La La La La La La
```

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 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)

The Format function

- The first argument passed to the format function is the item that you wish to format
- The second argument passed to the function is the formatting "pattern" you wish to apply to this item
- This pattern varies based on what you would like to do to the item in question
- Once the pattern is applied to the item the format function will return a string version of your item with the formatting pattern applied

String Formatting

- One common use of string formatting is to generate a string that contains a known # of characters
- For example, say you have the strings "Harry" and "Computer Science". You might want to generate output that looks like the following given these items:

```
Name        Department
Harry       Computer Science
```

- In this case we need to ensure that the strings "Name" and "Harry" are the same width so that the strings that come after them ("Department" and "Computer Science") line up correctly.

String Formatting

- You can use the format() function to "pad" a string with extra spaces at the beginning or the end of the string.
- For example:

```
x = format('Harry', '<20s')
```

- This will generate a new string (x) that contains the string 'Harry' plus 15 spaces at the end of the string. The total length of the string in this case will be 20 characters.
- 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

String Formatting

- You can also have Python right justify the string and place the spaces at the beginning of the new string by doing the following:

```
b = format('Harry', '>20s')
```

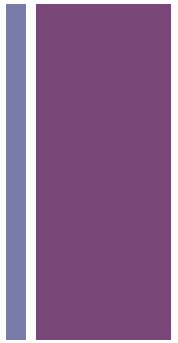
String Formatting

```
x = “Hello, World!”

y = format(x, ‘>20s’)

print (x)

>> Hello, World!

print (y)

>>          Hello, World!
```

Formatting Numbers

- 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



```
a = 1/6

print (a)                          0.16666666666666666


b = format (a, '.2f')              0.17

print (b)
```

Formatting patterns

```
a = 10000/6

b = format (a, '.2f')     # format a as a 2 digit float

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

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

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

Formatting Percentages

```
a = 0.52


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

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

print (format(a, '.0%'))     52%
```

Formatting Integers

```
a = 20000


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

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

Programming Challenge

- Write a program that generates the 2 times table, like this:

```
Number 1        Number 2        N1 * N2
2               1               2
2               2               4
2               3               6
2               4               8
```