PYTHON BOOT CAMP

Module 3: Math Functions, Strings, and Objects

Objectives

- To solve mathematics problems by using the functions in the **math** module (§3.2).
- To represent and process strings and characters (§§3.3-3.4).
- To encode characters using ASCII and Unicode (§§3.3.1-3.3.2).
- To use the **ord** to obtain a numerical code for a character and **chr** to convert a numerical code to a character (§3.3.3).
- To represent special characters using the escape sequence (§3.3.4).
- To invoke the **print** function with the end argument (§3.3.5).
- To convert numbers to a string using the **str** function (§3.3.6).
- To use the + operator to concatenate strings (§3.3.7).
- To read strings from the console (§3.3.8).
- To introduce objects and methods (§3.5).
- To format numbers and strings using the **format** function (§3.6).
- To draw various shapes (§3.7).
- To draw graphics with colors and fonts (§3.8).

Common Python Functions

- First, what is a function?
  - A function is a group of statements that performs a specific task
  - And we can broadly classify functions into:
    1. Those functions that we will fully define, write, and customize
       - This comes later (Chapter 6)
    2. Those functions that are prebuilt and available to the programmer as part of the library of the given language
  - Guess what?
    - We've been using functions since Day 1!
      - `eval`, `input`, `int`, and even `print`
      - These are all built-in functions and part of the Python library

Common Python Functions

- Some built-in math functions:

TABLE 3.1 Simple Python Built-in Functions

Function	*Description*	*Example*
`abs(x)`	Returns the absolute value for `x`.	`abs(-2)` is `2`
`max(x1, x2, ...)`	Returns the largest among `x1`, `x2`, ...	`max(1, 5, 2)` is `5`
`min(x1, x2, ...)`	Returns the smallest among `x1`, `x2`, ...	`min(1, 5, 2)` is `1`
`pow(a, b)`	Returns a^b. Same as `a ** b`.	`pow(2, 3)` is `8`

`round(x)`	Returns an integer nearest to `x`. If `x` is equally close to two integers, the even one is returned.	`round(5.4)` is `5`
		`round(5.5)` is `6`
		`round(4.5)` is `4`

`round(x, n)`	Returns the float value rounded to `n` digits after the decimal point.	`round(5.466, 2)` is `5.47` `round(5.463, 2)` is `5.46`

  - These functions are so common that no “imports” are needed in order for them to work…you just use them

Common Python Functions

- Some built-in math functions:

```
>>> abs(-3) # Returns the absolute value
3
>>> abs(-3.5) # Returns the absolute value
3.5
>>> max(2, 3, 4, 6) # Returns the maximum number
6
>>> min(2, 3, 4) # Returns the minimum number
2
>>> pow(2, 3) # Same as 2 ** 3
8
>>> pow(2.5, 3.5) # Same as 2.5 ** 3.5
24.705294220065465
>>> round(3.51) # Rounds to its nearest integer
4
>>> round(3.4) # Rounds to its nearest integer
3
>>> round(3.1456, 3) # Rounds to 3 digits after the decimal point
3.146
>>>
```

Common Python Functions

- Additional math functions:
  - The Python `math` module can be imported and provides additional math functions and some famous constants
    - Functions include:
      - `exp`, `sqrt`, `log`, `sin`, `cos`, and more
    - Constants include:
      - `PI` and `e`
    - You import the module similar to importing turtle
    - You simply type:
      ```
      import math
      ```
      at the beginning of your program

Common Python Functions

TABLE 3.2 Mathematical Functions

Function	*Description*	*Example*
`fabs(x)`	Returns the absolute value for `x` as a float.	`fabs(-2)` is `2.0`
`ceil(x)`	Rounds `x` up to its nearest integer and returns that integer.	`ceil(2.1)` is `3` `ceil(-2.1)` is `-2`
`floor(x)`	Rounds `x` down to its nearest integer and returns that integer.	`floor(2.1)` is `2` `floor(-2.1)` is `-3`
`exp(x)`	Returns the exponential function of `x` (e^x).	`exp(1)` is `2.71828`
`log(x)`	Returns the natural logarithm of `x`.	`log(2.71828)` is `1.0`
`log(x, base)`	Returns the logarithm of `x` for the specified base.	`log(100, 10)` is `2.0`
`sqrt(x)`	Returns the square root of `x`.	`sqrt(4.0)` is `2`
`sin(x)`	Returns the sine of `x`. `x` represents an angle in radians.	`sin(3.14159 / 2)` is `1` `sin(3.14159)` is `0`
`asin(x)`	Returns the angle in radians for the inverse of sine.	`asin(1.0)` is `1.57` `asin(0.5)` is `0.523599`
`cos(x)`	Returns the cosine of `x`. `x` represents an angle in radians.	`cos(3.14159 / 2)` is `0` `cos(3.14159)` is `-1`
`acos(x)`	Returns the angle in radians for the inverse of cosine.	`acos(1.0)` is `0` `acos(0.5)` is `1.0472`
`tan(x)`	Returns the tangent of `x`. `x` represents an angle in radians.	`tan(3.14159 / 4)` is `1` `tan(0.0)` is `0`
`degrees(x)`	Converts angle `x` from radians to degrees.	`degrees(1.57)` is `90`
`radians(x)`	Converts angle `x` from degrees to radians.	`radians(90)` is `1.57`

Common Python Functions

- Example program:

LISTING 3.1 MathFunctions.py

```
1  import math # import math module to use the math functions
2
3  # Test algebraic functions
4  print("exp(1.0) =", math.exp(1))
5  print("log(2.78) =", math.log(math.e))
6  print("log10(10, 10) =", math.log(10, 10))
7  print("sqrt(4.0) =", math.sqrt(4.0))
8
9  # Test trigonometric functions
10 print("sin(PI / 2) =", math.sin(math.pi / 2))
11 print("cos(PI / 2) =", math.cos(math.pi / 2))
12 print("tan(PI / 2) =", math.tan(math.pi / 2))
13 print("degrees(1.57) =", math.degrees(1.57))
14 print("radians(90) =", math.radians(90))
```

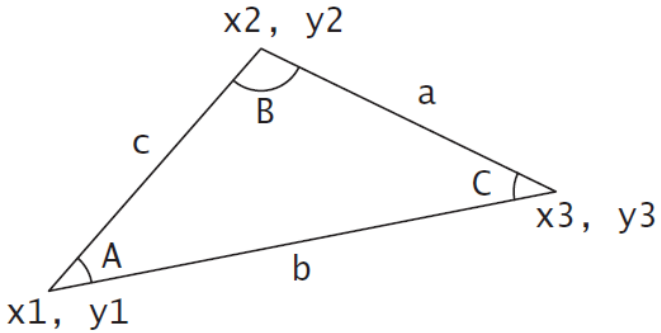
```
exp(1.0) = 2.71828182846
log(2.78) = 1.0
log10(10, 10) = 1.0
sqrt(4.0) = 2.0
sin(PI / 2) = 1.0
cos(PI / 2) = 6.12323399574e-17
tan(PI / 2) = 1.63312393532e+16
degrees(1.57) = 89.9543738355
radians(90) = 1.57079632679
```

Common Python Functions

- Example usage:
  - Having access to these math functions opens doors to solve a variety of computational problems
  - An important note:
    - Many are not comfortable with math
    - But that's no reason to be scared when seeing formulas!
    - You needn't derive the formula
    - All we need to know is what the formula does and how to use it
      - Just like we didn't "derive" how a car engine was put together
      - We just need to know what it does and how to use it

Common Python Functions

- Example usage:
  - For example, given three vertices of a triangle:



we can compute the three angles as follows:

```
A = acos((a * a - b * b - c * c) / (-2 * b * c))
B = acos((b * b - a * a - c * c) / (-2 * a * c))
C = acos((c * c - b * b - a * a) / (-2 * a * b))
```

- And with that knowledge, we can write a simple program…

Common Python Functions

- Example usage:

LISTING 3.2 ComputeAngles.py

```
 1  import math
 2
 3  x1, y1, x2, y2, x3, y3 = eval(input("Enter three points: "))
 4
 5  a = math.sqrt((x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3))
 6  b = math.sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3))
 7  c = math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2))
 8
 9  A = math.degrees(math.acos((a * a - b * b - c * c) / (-2 * b * c)))
10  B = math.degrees(math.acos((b * b - a * a - c * c) / (-2 * a * c)))
11  C = math.degrees(math.acos((c * c - b * b - a * a) / (-2 * a * b)))
12
13  print("The three angles are ", round(A * 100) / 100.0,
14      round(B * 100) / 100.0, round(C * 100) / 100.0)
```

```
Enter three points: 1, 1, 6.5, 1, 6.5, 2.5 ↵Enter
The three angles are 15.26 90.0 74.74
```

Common Python Functions

Check Point:

Evaluate the following functions:

(a) `math.sqrt(4)`

(b) `math.sin(2 * math.pi)`

(c) `math.cos(2 * math.pi)`

(d) `min(2, 2, 1)`

(e) `math.log(math.e)`

(f) `math.exp(1)`

(g) `max(2, 3, 4)`

(h) `abs(-2.5)`

(i) `math.ceil(-2.5)`

(j) `math.floor(-2.5)`

(k) `round(3.5)`

(l) `round(-2.5)`

(m) `math.fabs(2.5)`

(n) `math.ceil(2.5)`

(o) `math.floor(2.5)`

(p) `round(-2.5)`

(q) `round(2.6)`

(r) `round(math.fabs(-2.5))`

Strings and Characters

- What is a string?
  - A string is a sequence of characters
    - And this sequence could just be a string of numbers
    - Example:
      - **"3.145"** would be considered a string with five characters in it
  - In Python, a string must be enclosed in either double quotes (“) or single quotes (‘)
  - Examples:

```
message = “good morning”
letter = “A”
letter = ‘A’ # these are the same!
number_string = “2018” # same as number_string = ‘2018’
```

Strings and Characters

- What is a string?
  - Note:
    - Python does not have a specific data type for a single character
      - Many (most) languages do!
    - In Python, a single character is simply represented as a single-character string
  - We'd like to be consistent with other languages:
  - Therefore:
    - Double quotes will be used for a string with more than one character
    - Single quotes will be used for a single character string

A Brief Hiatus…

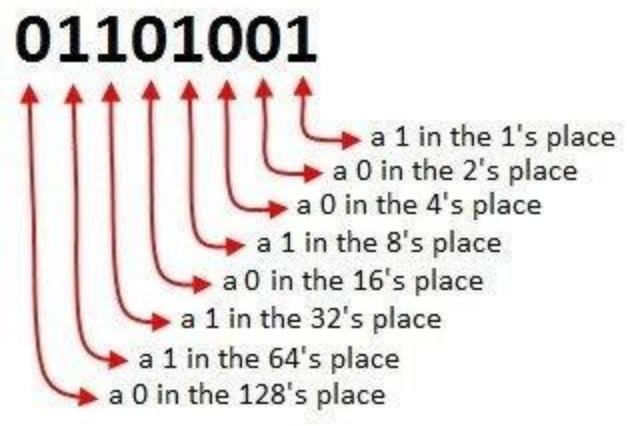
- A (very quick) primer on numbers!
  - The most common number system and the one we use most often is decimal
  - Decimal is base what?
    - Base 10
  - What does this mean?
    - Means there are ten numbers we use
    - 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
  - Computers use binary numbers. Binary is base what?
    - Base 2
  - What does this mean?
    - There are only two numbers used: 0 and 1 (known as bits)

A Brief Hiatus…

- A (very quick) primer on numbers!
  - How to convert from binary to decimal?
  - This is actually really easy
    - Each digit in a binary number can be 1 or 0
      - Think of this as on or off
    - And each of these digits has a value (a weight)
      - And that value counts towards the total if the bit is set to 1 (if it is “on”)
    - The least-significant bit is on the right
      - If the bit is 1, it’s value is simply 1
    - The values of each digit to the left increase by powers of 2
      - 2, 4, 8, 16, 32, 64, 128, 256, …
  - This is easiest to understand with pictures and examples…

A Brief Hiatus…

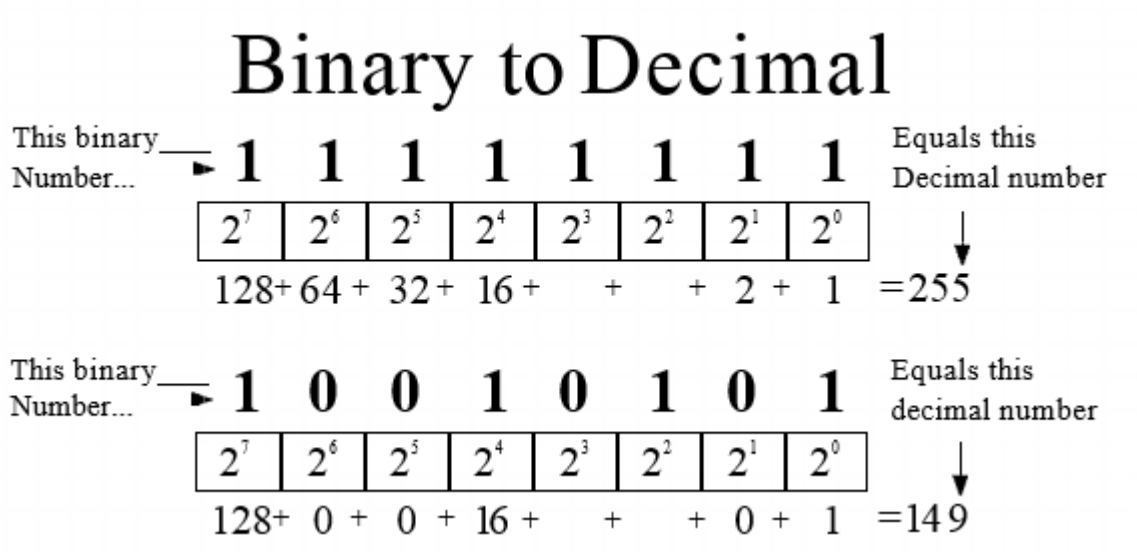
- A (very quick) primer on numbers!
  - Consider the following:



    - The decimal equivalent is:
      - 64 + 32 + 8 + 1 = 105

A Brief Hiatus…

- A (very quick) primer on numbers!
  - Another example/picture:



A Brief Hiatus…

- A (very quick) primer on numbers!
  - Check Yourself:
    What is the decimal value of the following:
    - 1001
      - 9
    - 1010
      - 10
    - 0111
      - 7
    - 1111
      - 15
    - 1000
      - 8

A Brief Hiatus…

- A (very quick) primer on numbers!
  - Binary is easy to understand (once we practice it)
  - But it's a pain to represent!
    - It takes so much digits to represent a basic number!
  - Hexadecimal to the rescue!
    - Hexadecimal (aka Hex) is another number system
    - Hex is base what?
      - Hex is base 16
    - What does this mean?
      - It means there are 16 numbers
      - Huh? But we only have 10 numbers (0 to 9). How do we get 16???
      - It is kinda weird at first, but here are the 16 hex numbers:
        - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
        - So F in hex is the same as 15 in decimal

A Brief Hiatus…

- A (very quick) primer on numbers!
  - So why hex?
    - Hex can very seamlessly (easily) represent binary numbers

Decimal (Base 10)	Binary (Base 2)	Hexadecimal (Base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Strings and Characters

- Character encoding:
  - Computer uses binary numbers internally
    - a sequence of 0s and 1s
  - All characters are in fact stored as a sequence of 0s and 1s
  - Mapping a character to its binary representation is called character encoding
  - There are different ways to map a character to binary
  - Two popular varieties:
    - ASCII
    - Unicode

Strings and Characters

- ASCII
  - Stands for:
    American Standard Code for Information Interchange
    - no need to remember that!
  - ASCII is a 7-bit encoding scheme
    - What does that mean?
    - This means that a total of 7 bits are used to represent characters
      - Example:
        - 0110110 is a group of 7 bits
          - It represents a specific character in ASCII
        - 1100100 is another group of 7 bits
          - And it represents a different character in ASCII
    - With 7 bits, this means 2^7 possible different groups of those bits
    - So 128 different characters can be encoded with ASCII

Strings and Characters

- ASCII
  - 128 characters is not very much
    - Sure, it may suffice a single language, but not much more
    - Consider 26 uppercase characters in English alphabet
    - And 26 lower case
      - Yes, these are encoded differently…they are different characters
    - That's already 52 characters
    - Now add in numbers, punctuation marks, and other common characters
    - We quickly use up those 128 spots in ASCII
  - Long story short:
    - ASCII simply isn't enough
    - So enter Unicode…

Strings and Characters

- Unicode
  - Easy summary:
    - Allows encoding of 1,114,112 characters!
    - Yes, MORE than sufficient for everything we need
  - Encoding:
    - Unicode starts with \u and then has 4 hexadecimal digits
    - These digits run from \u0000 to \uFFFF
  - Example:

LISTING 3.3 DisplayUnicode.py

```
1  import turtle
2
3  turtle.write("\u6B22\u8FCE \u03b1 \u03b2 \u03b3")
4
5  turtle.done()
```

Strings and Characters

- **`ord`** and **`chr`** functions:
  - Python provides two helpful functions:
    - **`ord(ch)`** function for returning the ASCII code for the character **`ch`**
    - **`chr(code)`** function for returning the character represented by the code.

```
>>> ch = 'a'
>>> ord(ch)
97
>>> chr(98)
'b'
>>> ord('A')
65
>>>
```

Strings and Characters

- **Escape Sequences for Special Characters**
  - Consider the following example:
    - How would you print a message with quotation marks in Python?
    - Meaning, what if you wanted to quote someone and print the actual quotation
    - Could you do this?

```
print("He said, "John's program is easy to read"")
```

    - The answer is no. That won't work
    - When Python sees the second double quotation mark, it understands that the string is finished/complete
    - You would print this as follows:

```
print("He said, \"John's program is easy to read\"")
```

    - Notice the backslashes…that is called an **ESCAPE sequence**

Strings and Characters

- Escape Sequences for Special Characters
  - Escape sequences is a special notation used to represent special characters
    - This notation consists of a backslash followed by a letter or a combination of digits

TABLE 3.3 Python Escape Sequences

Character Escape Sequence	*Name*	*Numeric Value*
`\b`	Backspace	8
`\t`	Tab	9
`\n`	Linefeed	10
`\f`	Formfeed	12
`\r`	Carriage Return	13
`\\`	Backslash	92
`\'`	Single Quote	39
`\"`	Double Quote	34

Strings and Characters

- Printing without the newline
  - The print function automatically prints a new line (\n)
    - This causes the output to advance to the next line
  - What if you don't want to advance to the next line?
    - You use the print function with a special argument

```
print(item, end = "anyendingstring")
```

  - For example, consider the following code:

```
print("AAA", end = ' ')
print("BBB", end = '')
print("CCC", end = '***')
print("DDD", end = '***')
```

    - Output: **AAA BBBCCC***DDD*****

Strings and Characters

- Printing without the newline
  - Another example:
    - Consider the following code:

```
radius = 3
print("The area is", radius * radius * math.pi, end = ' ')
print("and the perimeter is", 2 * radius)
```

    - The output:

```
The area is 28.26 and the perimeter is 6
```

Strings and Characters

- The `str` function
  - The str function can be used to convert a number into a string

```
>>> s = str(3.4) # Convert a float to string
>>> s
'3.4'
>>> s = str(3) # Convert an integer to string
>>> s
'3'
>>>
```

Strings and Characters

- The String Concatenation Operator
  - We normally view the + sign as addition
    - and this is okay
  - But in programming languages, the + operator has another meaning: concatenation
  - We can use the + operator to concatenate two strings

```
1  >>> message = "Welcome " + "to " + "Python"
2  >>> message
3  'Welcome to Python'
4  >>> chapterNo = 3
5  >>> s = "Chapter " + str(chapterNo)
6  >>> s
7  'Chapter 3'
8  >>>
```

Strings and Characters

- Reading strings from the console
  - We've actually been doing this for some time now
  - Python understands all input as a string
    - We then used the eval and int functions to convert the string to other values
  - Example:

```
s1 = input("Enter a string: ")
s2 = input("Enter a string: ")
s3 = input("Enter a string: ")
print("s1 is " + s1)
print("s2 is " + s2)
print("s3 is " + s3)
```

```
Enter a string: Welcome ↵Enter
Enter a string: to ↵Enter
Enter a string: Python ↵Enter
s1 is Welcome
s2 is to
s3 is Python
```

Problem 1: Range Calculator

- Write a program to calculate the number of miles remaining before you run out of gas!
  - This is very common in most cars these days.
- Remember:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

Problem 1: Range Calculator

- Write a program to calculate the number of miles remaining before you run out of gas!
  - This is very common in most cars these days.
- Step 1: Problem-solving Phase
  - First, let us see a sample run of the program…

```
>>> %Run milesLeft.py
 What is the initial odometer reading:  50000
 How many gallons of gas does your tank hold:  15
 What was your second odometer reading:  50020
 How many gallons were left then:  14
 You can go 280 miles before needing to refuel.
 280.0

>>>
```

Problem 1: Range Calculator

- Write a program to calculate the number of miles remaining before you run out of gas!
  - This is very common in most cars these days.
- Step 1: Problem-solving Phase
  - After some thought (few minutes probably)…
  - Hopefully we realize the following:
    - We need the miles driven
      - Ending reading – starting reading
    - We need the amount of gas used
      - Starting gas – ending gas
    - We need to calculate the miles per gallon thus far
    - And we then multiply that times the gas remaining…

Problem 1: Range Calculator

- Write a program to calculate the number of miles remaining before you run out of gas!
  - This is very common in most cars these days.
- Step 2: Implementation Phase
  - Check portal for a sample solution!

Intro to Objects and Methods

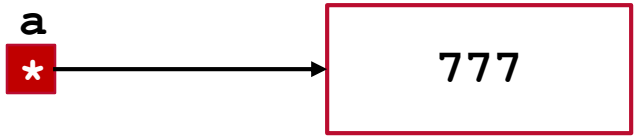
- What are objects and OOP?
  - OOP stands for Object-oriented Programming
    - For now…that's all you need to know!
    - We'll get to that concept later on
  - At the core of OOP is objects…so what are objects?
  - Well, in Python, ALL data are objects!
    - This includes numbers and strings
    - And this is different from many other languages

Intro to Objects and Methods

- What are objects and OOP?
  - "Um…again, so what are objects?!?"
    - Consider and `int` variable such as `a = 777`
      - normally students would imagine that int value, 777, just floating around in computer memory
      - With objects, we do not think of variable **a** as storing the value 777
      - Rather, **a** stores a reference, and that reference points to a box
        - and the value 777 can be found inside that box
  - A picture is helpful
    - Given the code:

      ```
      a = 777
      ```

    - Here's how you can visualize this:



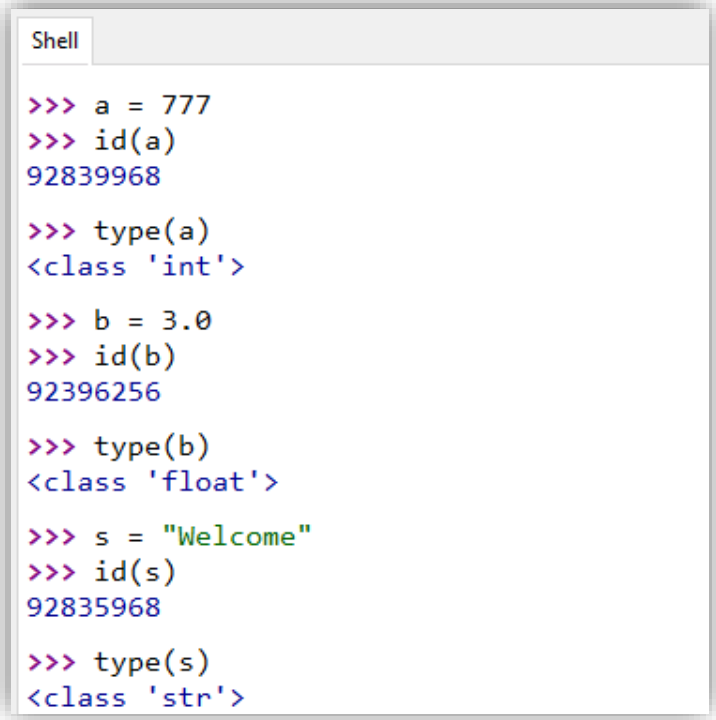
Intro to Objects and Methods

- What are objects and OOP?
  - "So why is this helpful?"
    - Long answer: many reasons…and they will come up
    - But for now, we can perform operations on these objects!
      - Operations made for and used with objects are called **methods**.
  - Example:
    - `s = "Welcome"`
      - Remember: the variable `s` stores a reference that points to a box, and the string "Welcome" can be found inside that box
      - Now we can perform methods on that box!

```
s1 = s.lower()
print(s1) # "welcome" is printed
s2 = s.upper()
print(s2) # "WELCOME" is printed
```

Intro to Objects and Methods

- Some interesting functions for objects
  - Python gives the `id` and the `type` functions to get information about our objects
    - `id`: this is the actual reference saved inside the variable
      - Such as the reference saved in "`a`" on that last picture
    - `type`: this refers to the type of the given object
    - These functions are rarely used in programming
    - But they are helpful when first learning about objects



Intro to Objects and Methods

- Additional useful String methods
  - strip():
    - used to removed whitespace characters from both sides of a string
      - Whitespace includes spaces, tabs, and newlines

```
Shell
>>> s = "\t\t\t\t\tHello           "
>>> print(s)
                                        Hello

>>> s.strip()
'Hello'
>>> print(s)
                                        Hello

>>> s = s.strip()
>>> print(s)
 Hello
```

  - You can read about other methods here:
    - https://www.w3schools.com/python/python_ref_string.asp

Formatting Numbers & Strings

- Formatting is often helpful and even needed
  - Consider the following code:

```
>>> amount = 12618.98
>>> interestRate = 0.0013
>>> interest = amount * interestRate
>>> print("Interest is", interest)
Interest is 16.404674
>>>
```

    - The interest is currency
      - So it's desirable to have two decimals
  - We could rewrite the code as follows

```
>>> amount = 12618.98
>>> interestRate = 0.0013
>>> interest = amount * interestRate
>>> print("Interest is", round(interest, 2))
Interest is 16.4
>>>
```

Should be 16.40

That's still not correct!

Formatting Numbers & Strings

- Formatting is often helpful and even needed
  - The solution is formatted printing:

```
>>> amount = 12618.98
>>> interestRate = 0.0013
>>> interest = amount * interestRate
>>> print("Interest is", format(interest, ".2f") )
Interest is 16.40
>>>
```

  - Syntax:

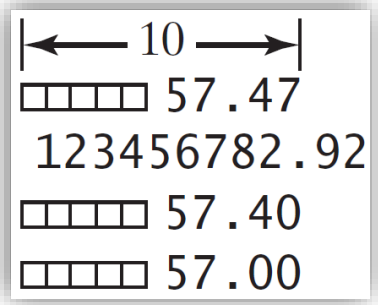
    `format(item, format-specifier)`

    - Here, `item` is a number of a string
    - And `format-specifier` is a string that specifies how the item is to be formatted

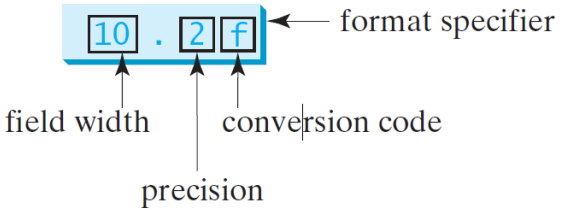
Formatting Numbers & Strings

- Formatting Floating-Point Numbers
  - Consider the following code and output:

```
print(format(57.467657, "10.2f"))
print(format(12345678.923, "10.2f"))
print(format(57.4, "10.2f"))
print(format(57, "10.2f"))
```



  - You specify a width, a precision, and a conversion code
    - The width is how many spaces to print the number
    - The precision is how many digits after the decimal place
    - The conversion code, in this example, **f**, tells Python that we are formatting a **f**loating-point number



Formatting Numbers & Strings

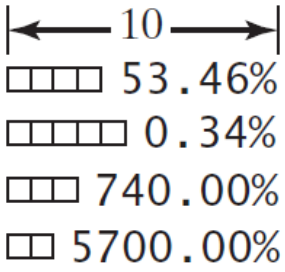
- Formatting Floating-Point Numbers
  - Comments:
    - By default, the number is aligned to the right within the specified width
    - If the number is larger than the width, the width is automatically increased
    - You can also omit the width specifier
      - Example: `print(format(57.467657, ".2f"))`
      - In this case, the width is set automatically

Formatting Numbers & Strings

- Formatting as a Percentage
  - We can use conversion code % to format a percentage
    - And if we use "10.2%" as the full format specifier, the number is first multiplied by 100 and displayed with a % sign

```
print(format(0.53457, "10.2%"))
print(format(0.0033923, "10.2%"))
print(format(7.4, "10.2%"))
print(format(57, "10.2%"))
```

    - Result:

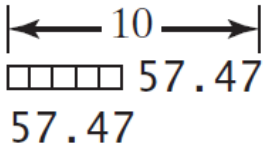


Formatting Numbers & Strings

- Justifying Format
  - Default: number is right justified
  - We can use the < or > symbols for justification as well

```
print(format(57.467657, "10.2f"))
print(format(57.467657, "<10.2f"))
```

displays

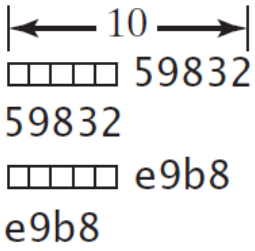


Formatting Numbers & Strings

- Formatting Integers
  - The conversion codes **`d`**, **`x`**, **`o`**, and **`b`**:
    - used to format an integer in decimal, hexadecimal, octal, or binary
    - We can also specify a width for the conversion

```
print(format(59832, "10d"))
print(format(59832, "<10d"))
print(format(59832, "10x"))
print(format(59832, "<10x"))
```

displays



Formatting Numbers & Strings

- Formatting Strings
  - You can use the conversion code **s** to format a string with a specified width

```
print(format("Welcome to Python", "20s"))
print(format("Welcome to Python", "<20s"))
print(format("Welcome to Python", ">20s"))
print(format("Welcome to Python and Java", ">20s"))
```

displays


```
Welcome to Python
Welcome to Python
   Welcome to Python
Welcome to Python and Java
```

Formatting Numbers & Strings

- Frequently Used Specifiers

Specifier	*Format*
`"10.2f"`	Format the float item with width 10 and precision 2.
`"10.2e"`	Format the float item in scientific notation with width 10 and precision 2.
`"5d"`	Format the integer item in decimal with width 5.
`"5x"`	Format the integer item in hexadecimal with width 5.
`"5o"`	Format the integer item in octal with width 5.
`"5b"`	Format the integer item in binary with width 5.
`"10.2%"`	Format the number in decimal.
`"50s"`	Format the string item with width 50.
`"<10.2f"`	Left-justify the formatted item.
`">10.2f"`	Right-justify the formatted item.

Problem 2: Kool-Aid

- Write a program to determine the number of cups of Kool-Aid that must be sold in order to meet a specified goal (see sample).
- Remember:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

Problem 2: Kool-Aid

- Write a program to determine the number of cups of Kool-Aid that must be sold in order to meet a specified goal (see sample).
- Step 1: Problem-solving Phase
  - First, let us see a sample run of the program…

```
>>> %Run koolaid.py
 How many dollars is the rent for your stand?  5
 How many cents do the materials cost, per glass?  2
 How many cents do you charge per glass?  25
 What is your profit goal, in dollars?  50
 You must sell 240 cups of Kool-Aid to meet your goal.

>>>
```

Problem 2: Kool-Aid

- Write a program to determine the number of cups of Kool-Aid that must be sold in order to meet a specified goal (see sample).
- Step 1: Problem-solving Phase
  - Spend some time to think this one through on paper
  - Once you have it solved on paper, try to code it
  - You'll likely get really close
    - and maybe exactly close on some cases
  - But there's one additional thing to think of…

Problem 2: Kool-Aid

- Write a program to determine the number of cups of Kool-Aid that must be sold in order to meet a specified goal (see sample).
- Step 2: Implementation Phase
  - Check portal for a sample solution!

PYTHON BOOT CAMP

Module 3: Math Functions, Strings, and Objects