Introduction to Problem Solving in Python

COSI 10A

Class objectives

- Math module
- Review

Review: Multiple parameters

```
def main():
    print_number(4, 9)
    print_number(17, 6)
    print_number(8, 0)
    print_number(0, 8)

def print_number(number, count):
    for i in range(0, count):
        print(number, end="")
    print()

main()
```

Output:

```
444444444
171717171717


00000000
```

Review: Value semantic

- When numbers and strings are passed as parameters, their values are copied
- Modifying the parameter will not affect the variable passed in

Review: Returning a value

Syntax:

```
def name(parameters):
    statements
    …
    return expression
```

- When Python reaches a return statement
  - It evaluates the expression
  - It substitutes the return value in place of the call
  - It goes back to the caller and continues after the method call

Review: Common error: not storing

- Many students incorrectly think that a `return` statement sends a variable's name back to the calling method

```
def slope(x1, x2, y1, y2):
    dy = y2 - y1
    dx = x2 - x1
    result = dy / dx
    return result

def main():
    slope(0, 0, 6, 3)
    print("The slope is", result)  # ERROR: cannot find symbol: result

main()
```

Review: Fixing the error

- Returning sends the variable's *value* back. **Store the returned value into a variable** or use it in an expression

```
def slope(x1, x2, y1, y2):
    dy = y2 - y1
    dx = x2 - x1
    result = dy / dx
    return result

def main():
    x = slope(0, 0, 6, 3)
    print("The slope is", x)

main()
```

Review: Fixing the error

- Returning sends the variable's *value* back. Store the returned value into a variable **or use it in an expression**

```
def slope(x1, x2, y1, y2):
    dy = y2 - y1
    dx = x2 - x1
    result = dy / dx
    return result

def main():
    print("The slope is", slope(0, 0, 6, 3))

main()
```

Math

Math built-in functions

- Python has many useful built-in functions that represent mathematical operations
- These functions are part of a library known as `math` library
- Each library available is called a **module**

- **A module** is an individual unit of Python library functionality with a name
- To use a a library in your program you must place an **import statement** at the top of your program

Syntax:

```
import module
```

Example:

```
import math
```

Python’s Math module

Function name	**Description**	**Example**	**Result**
`math.ceil(`*value*`)`	rounds up	`math.ceil(2.13)`	`3.0`
`math.floor(`*value*`)`	rounds down	`math.floor(2.93)`	`2.0`
`math.sqrt(`*value*`)`	square root	`math.sqrt(2)`	`1.414213562…`

`math.sinh(`*value*`)`
`math.cosh(`*value*`)`
`math.tanh(`*value*`)`
	sine/cosine/tangent of an angle in radians		

`math.degrees(`*value*`)`
`math.radians(`*value*`)`
	convert degrees to radians and back		

`import math` is necessary to use the **above** functions

Function name	**Description**	**Example**	**Result**
`abs(`*value*`)`	absolute value	`abs(-308)`	`308`
`min(`*value1, value2*`)`	smaller of two values	`min(7, 2, 4, 3)`	`2`
`max(`*value1, value2*`)`	larger of two values	`max(11, 8)`	`11`

`round(`*value*`)`	nearest whole number	`round(3.647)`
		`round(3.647, 1)`
			`4`
			`3.7`

Built-in global functions

Python’s Math module

- The math module also defines some constants for frequently used mathematical values

Constant	Description
`math.e`	2.7182818...
`math.pi`	3.1415926...

`import math` is necessary

No output?

- Simply calling these functions produces no visible result

```
math.sqrt(81)    # no output
```

- Math functions: they all return a value
- The program runs the function computes the answer, and then “replaces” the call with its computed result value
- To see the result, we must print it or store it in a variable

```
result = math.sqrt(81)
print(result)                 # 9.0
```

Why return and no print?

- It might seem more useful for the `math` functions to print their results rather than returning them. Why don't they?
- Returning is more flexible than printing
- We can compute several things before printing:

```
sqrt1 = math.sqrt(100)
sqrt2 = math.sqrt(81)
print("Powers are", sqrt1, "and", sqrt2)
```

- We can combine the results of many computations:

```
k = 13 * math.sqrt(49) + 5 - math.ceil(17.8)
```

Quirks of real number

- Float values print too many digits

```
result = 1.0 / 3.0
print(result)         # 0.3333333333333
```

```
print(0.1 + 0.2)
```

Instead of 0.3, the output is `0.30000000000000004`

Type casting

- The general template for converting one type to another is:

Syntax:

```
type(expression)
```

Example:

```
int(4.75)               #convert to int 4
int(17.3)               #convert to 17
int(3.14159)            #convert to 3
float(42)               #convert to float 42.0
```

Example

- Write a function that calculates the hypotenuse of a triangle

```
import math

def hypotenuse(a, b):
    c = math.sqrt(a ** 2 + b ** 2)
    return c

def main():
    print("hypotenuse 1 =", hypotenuse(5, 12))
    result = hypotenuse(3, 4)
    print("hypotenuse 2 =", result)

main()
```

Function Signature: the name of a function, along with its number of parameters

Example

- Write a function that calculates the hypotenuse of a triangle

```
import math

def hypotenuse(a, b):
    c = math.sqrt(a ** 2 + b ** 2)
    return c

def main():
    print("hypotenuse 1 =", hypotenuse(5, 12))
    result = hypotenuse(3, 4)
    print("hypotenuse 2 =", result)

main()
```

Formal parameter: a variable that appears inside parentheses in the header of a function

Actual parameter: a specific value or expression that appears inside parentheses in a function call

Example

- Write a function that calculates the hypotenuse of a triangle

```
import math

def hypotenuse(a, b):
    c = math.sqrt(a ** 2 + b **2)
    return c

def main():
    print("hypotenuse 1 =", hypotenuse(5, 12))
    result = hypotenuse(3, 4)
    print("hypotenuse 2 =", result)

main()
```

- Do something with the value that is returned
- Print it, store it in a variable, or use it as part of a larger expression
- No statements after a return statement

Returning multiple values

- As a function can accept multiple parameters, it can also return multiple values
- **This is a feature of Python, most programming languages return only a single value**

```
def add_diff(a, b):
    add = a + b
    diff = a - b
    return add, diff

def main():
 res1, res2 = add_diff(6, 15)
    print("The sum is = ", res1, "The difference is = ", res2)
main()
```

ASCII art: Top half/w constant

```
|      <><>      |
|    <>....<>    |
|  <>........<>  |
```

```
SIZE = 3;

# Prints the expanding pattern of <> for the top half of the figure.
def top_half():
    for line in range(1, SIZE+1):
        print("|", end="")
        for space in range(line * -2 + (2*SIZE)):
           print(" ", end="")
        print("<>", end="")
        for dot in range(line * 4 - SIZE-1):
            print(".", end="")
        print("<>", end="")
        for space in range(line * -2 + (2*SIZE)):
            print(" ", end="")
        print("|")

top_half()
```

Parameters vs. Constants

- Constants are useful to increase the flexibility of a program. You can easily modify a program to behave differently
  - The major limitation is that the constant can change only when modified by the programmer
- Parameters are more flexible; you can specify the value to be used each time you call a function
- Use constants when you only want to change the value from execution to execution
- Use parameters if you want to use different values within a single execution

Optional parameters

- Write a function that produces the following output

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

Optional parameters

- Write a function that produces the following output

```
    *
   ***
  *****
 *******
*********
```

```
def draw_hat(size):
    for line in range(1, size):
        print(" " * (size - line), end ="")
        print("*" * (line * 2 -1))

def main():
    draw_hat(6)

main()
```

- In Python you can write a single function that accepts optional parameters
- An **optional parameter** is one for which the caller can pass an explicit value or can omit the parameter to receive a default value instead

Optional parameters

- Write a functions that produce the following output

```
def draw_hat(size = 6):
    for line in range(1, size):
        print(" " * (size - line), end ="")
        print("*" * (line * 2 -1))

def main():
    draw_hat(11) # pass size 11
    draw_hat()   # default size of 6

main()
```

```
          *
         ***
        *****
       *******
      *********
     ***********
    *************
   ***************
  *****************
 *******************
     *
    ***
   *****
  *******
 *********
```