Introduction to Problem Solving in Python

COSI 10A

Class objectives

- Parameters
- Value semantics

Review: Constants

- A **constant** is a fixed value visible to the whole program (global scope)
  - The value should only be set at declaration; shouldn't be reassigned

Syntax: **name = value**

name is usually in ALL_UPPER_CASE

Examples:

```
DAYS_IN_WEEK = 7
INTEREST_RATE = 3.5
SSN = 658234569
```

Review: Declaring a parameter

- Declaring a parameter means stating that a function requires a parameter in order to run

Syntax:

```
def <name> (<name>):
    <statement>(s)
```

Example:

```
def say_password(code):
    print("The password is:", code)
```

- When `say_password` is called, the caller <u>must</u> specify the **code** to print

Review: Passing a parameter

- Passing a parameter means calling a function specifying the value for its parameters

Syntax: ***<name>*** (***<expression>***)

Example:

```
say_password(42)
say_password(12345)
```

Output:

```
The password is 42
The password is 12345
```

Review: Passing a parameter

```
def main():
    say_password(3342)
    print()
    say_password("abracadabra")
    print()
    say_password(89.6)


def say_password(code):
    print("The password is:", code)

main()
```

Parameters and loops

- A parameter can guide the number of repetitions of a loop

Example:

```
chant(3)

def chant(times):
    for i in range(0, times):
        print("Just a salad...")
```

Output:

```
Just a salad...
Just a salad...
Just a salad...
```

Common errors

- If a function accepts a parameter, it is illegal to call it without passing any value for that parameter

```
chant()          # ERROR: parameter value required
```

- The value passed to a function must be of a type that will work

```
chant(3.7)       # ERROR: must be of type int if it
                 #        is used as a range bound
```

Multiple parameters

- A function can accept multiple parameters (separate by ,)
- When calling the function, you must pass values for each parameter

Declaration:

```
def <name>(<name>, ..., <name>):
    <statement>(s)
```

Call:

```
<name>(<exp>, <exp>, ..., <exp>)
```

Multiple parameters example

```
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
```

Parameters – Value semantics

Parameter Mystery

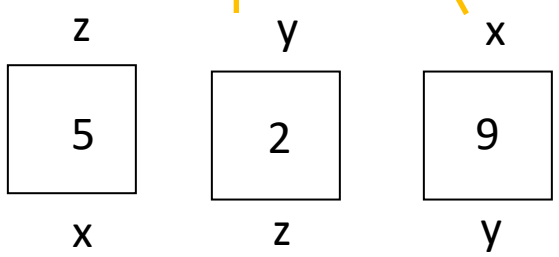
```
def main():
    x = 9
    y = 2
    z = 5

    mystery(z, y, x)

    mystery(y, x, z)

def mystery(x, z, y):
    print(z, "and", (y - x))

main()
```

Parameter Mystery

```
def main():
    x = 9
    y = 2
    z = 5

    mystery(z, y, x)

    mystery(y, x, z)

def mystery(x, z, y):
    print(z, "and", (y - x))

main()
```

Output:

```
2 and 4
9 and 3
```



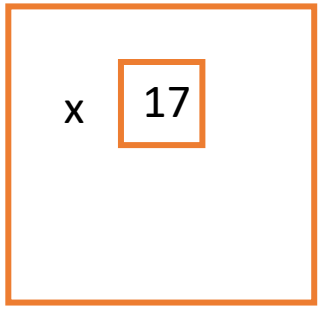
inside main

inside mystery function

Value semantic

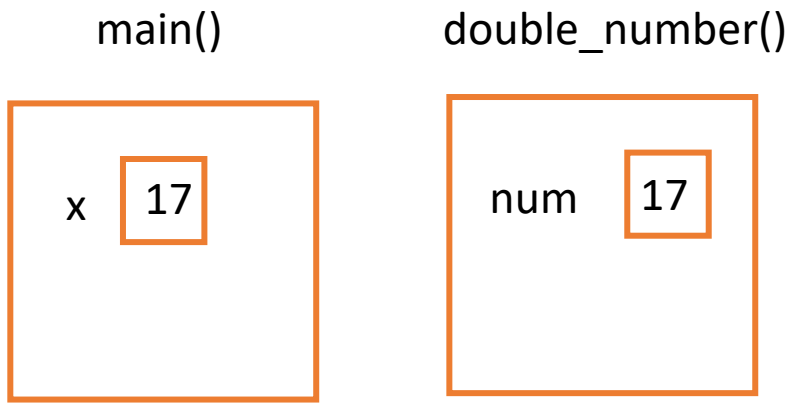
```
def double_number(num):
        print("in double_number, initial value = ", num)
        num = num * 2
        print("in double_number, final value = ", num)

def main():
        x = 17
        double_number(x)
        print("in main, x = ", x)
        print()

        num = 42
        double_number(num)
        print("in main, num = ", num)
        print()

main()
```

main()



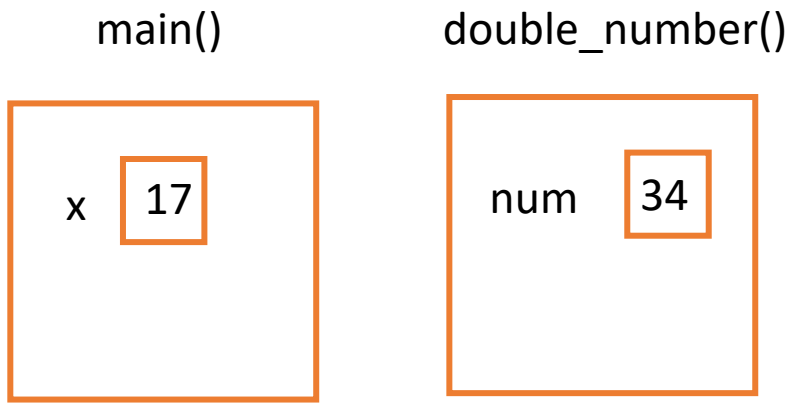
Value semantic

```
def double_number(num):
        print("in double_number, initial value = ", num)
        num = num * 2
        print("in double_number, final value = ", num)

def main():
        x = 17
        double_number(x)
        print("in main, x = ", x)
        print()

        num = 42
        double_number(num)
        print("in main, num = ", num)
        print()

main()
```



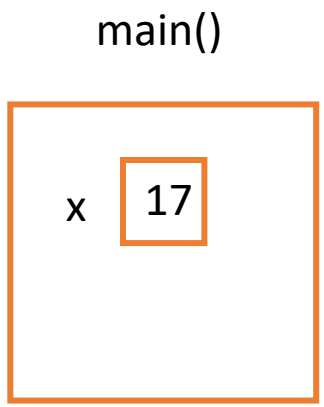
Value semantic

```
def double_number(num):
        print("in double_number, initial value = ", num)
        num = num * 2
        print("in double_number, final value = ", num)

def main():
        x = 17
        double_number(x)
        print("in main, x = ", x)
        print()

        num = 42
        double_number(num)
        print("in main, num = ", num)
        print()

main()
```



Value semantic

```
def double_number(num):
        print("in double_number, initial value = ", num)
        num = num * 2
        print("in double_number, final value = ", num)

def main():
        x = 17
        double_number(x)
        print("in main, x = ", x)
        print()

        num = 42
        double_number(num)
        print("in main, num = ", num)
        print()

main()
```



Print in `main` the value of x which is still 17

Value semantic

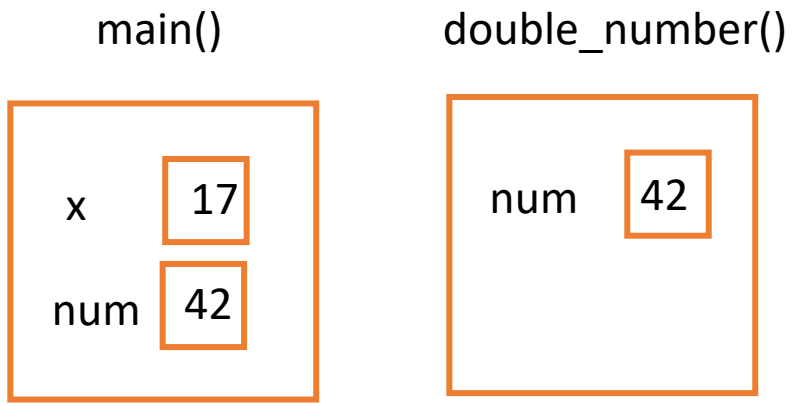
```
def double_number(num):
        print("in double_number, initial value = ", num)
        num = num * 2
        print("in double_number, final value = ", num)

def main():
        x = 17
        double_number(x)
        print("in main, x = ", x)
        print()

        num = 42
        double_number(num)
        print("in main, num = ", num)
        print()

main()
```

main()

x	17
num	42

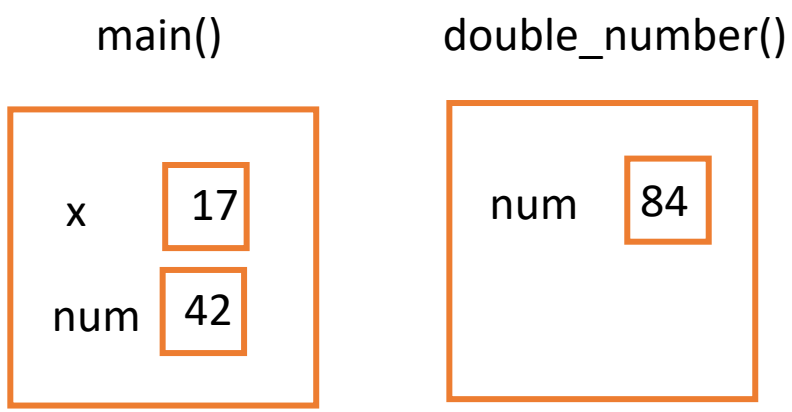
Value semantic

```
def double_number(num):
        print("in double_number, initial value = ", num)
        num = num * 2
        print("in double_number, final value = ", num)

def main():
        x = 17
        double_number(x)
        print("in main, x = ", x)
        print()

        num = 42
        double_number(num)
        print("in main, num = ", num)
        print()

main()
```



Value semantic

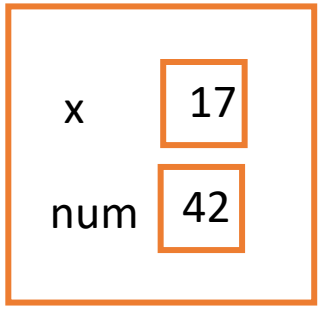
```
def double_number(num):
        print("in double_number, initial value = ", num)
        num = num * 2
        print("in double_number, final value = ", num)

def main():
        x = 17
        double_number(x)
        print("in main, x = ", x)
        print()

        num = 42
        double_number(num)
        print("in main, num = ", num)
        print()

main()
```



Value semantic

```
def double_number(num):
        print("in double_number, initial value = ", num)
        num = num * 2
        print("in double_number, final value = ", num)

def main():
        x = 17
        double_number(x)
        print("in main, x = ", x)
        print()

        num = 42
        double_number(num)
        print("in main, num = ", num)
        print()

main()
```

main()



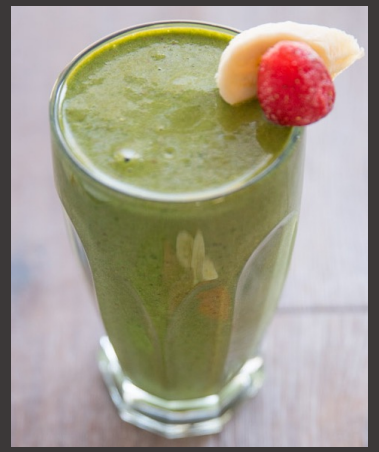
Print in `main` the value of num which is still 42

Value semantic

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

Returning values


NUTRiBULLET
MAGIC BULLET



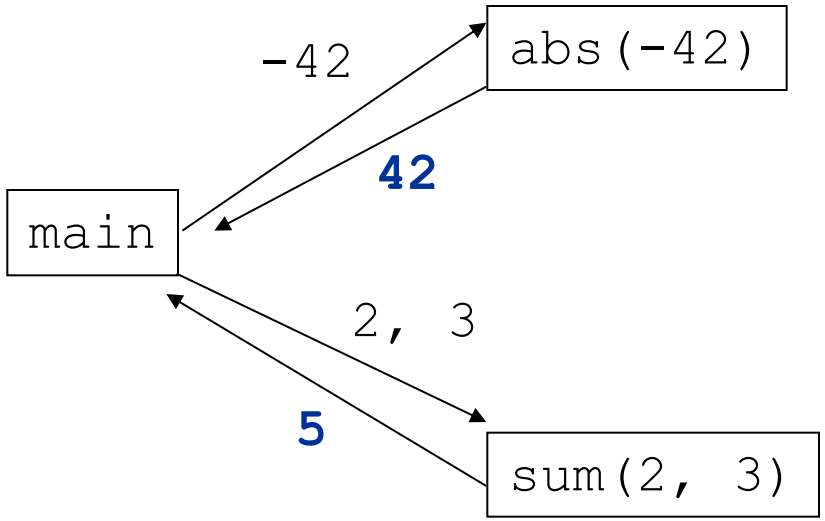
Returning values

- So far our functions they have been “action-oriented”, e.g. *print a box of stars*
- We now want to be able to write functions that compute values, e.g. *what do you get when you raise 3 to the power of 4?*
  - Your function can print the result to the console, but a better solution is to have your function use the result as an expression or a value

- Your function should **return** a value

Return

- **Return** means to send a value out as the result of a function
- Return values send information **out** from a function to its caller, as opposite to parameters that send values **into** a function



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

Returning examples

```
# Converts degrees Fahrenheit to Celsius.
def f_to_c(degrees_f):
    degrees_c = 5.0 / 9.0 * (degrees_f - 32)
    return degrees_c

def main():
    x = f_to_c(72)
    print(x)

main()
```

- You can shorten your function by returning an expression

```
# Converts degrees Fahrenheit to Celsius.
def f_to_c(degrees_f):
    return 5.0 / 9.0 * (degrees_f - 32)
```

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()
```

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()
```

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()
```