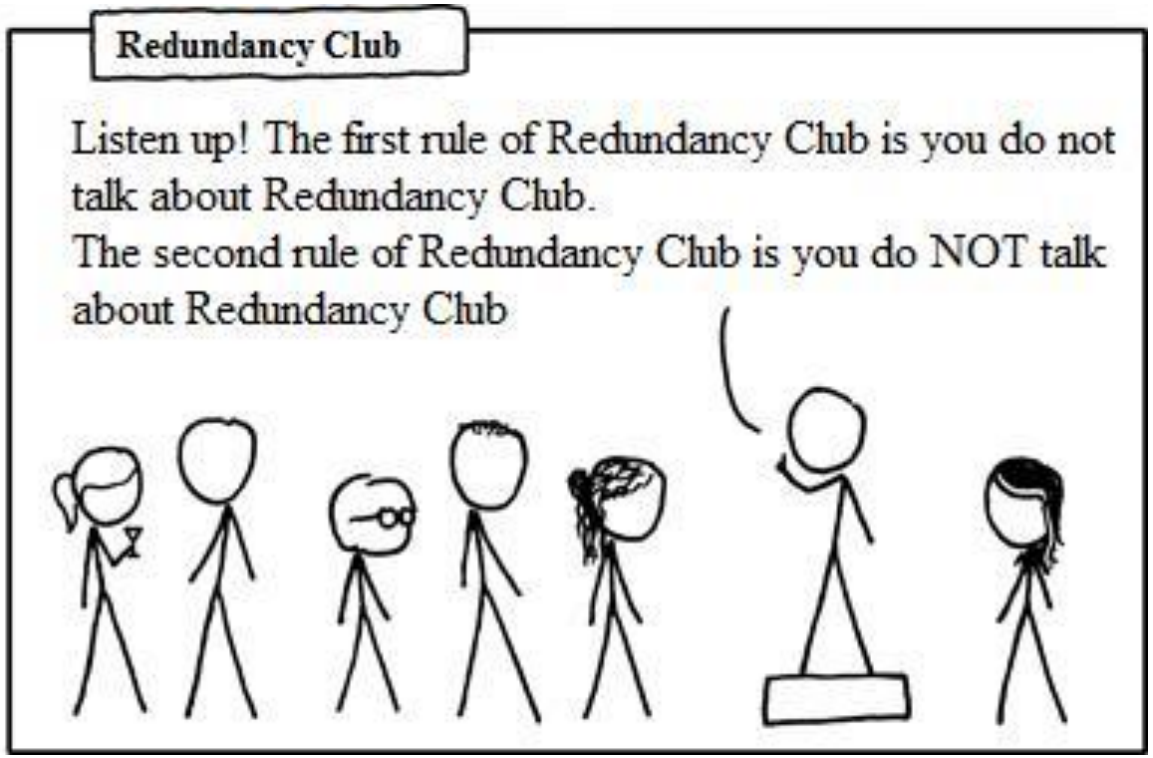
CSc 110, Spring 2017

Lecture 5: Constants and Parameters

Adapted from slides by Marty Stepp and Stuart Reges



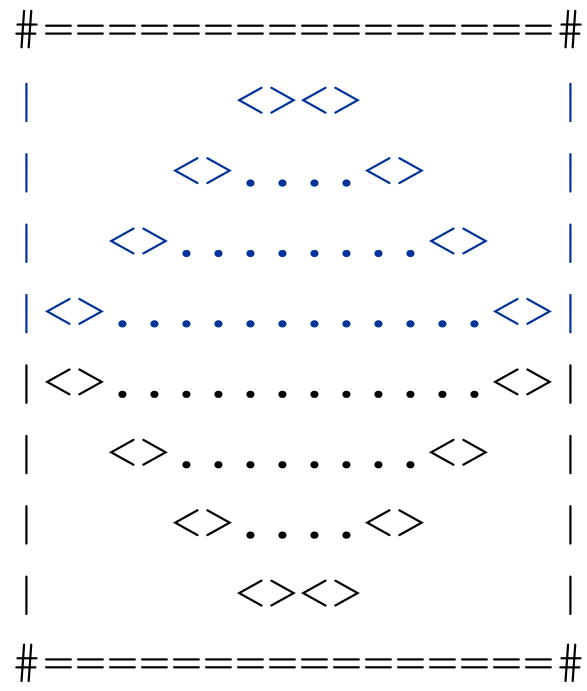
Pseudocode algorithm

1. Line
   - # , 16 =, #

2. Top half
   - |
   - spaces (decreasing)
   - <>
   - dots (increasing)
   - <>
   - spaces (same as above)
   - |

3. Bottom half (top half upside-down)

4. Line
   - # , 16 =, #



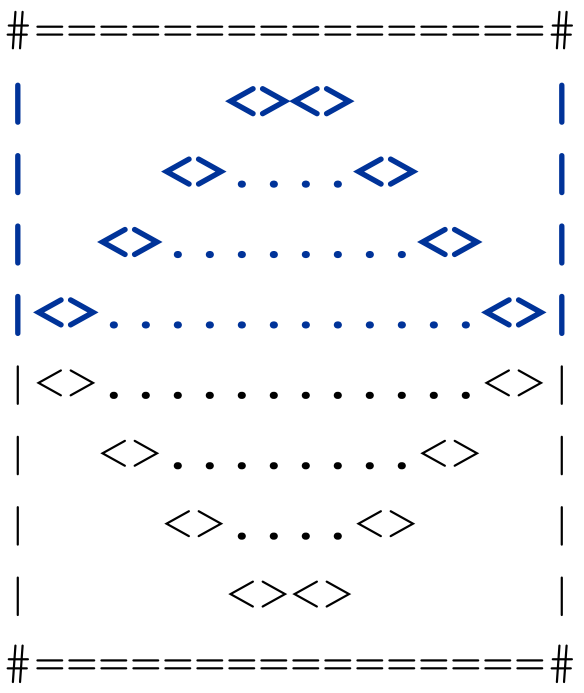
Functions from pseudocode

```
def main():
    line()
    top_half()
    bottom_half()
    line()

def top_half():
    for line in range(1, 5):
        # contents of each line

def bottom_half() {
    for line in range(1, 5):
        # contents of each line

def line():
        # ...
```

2. Tables

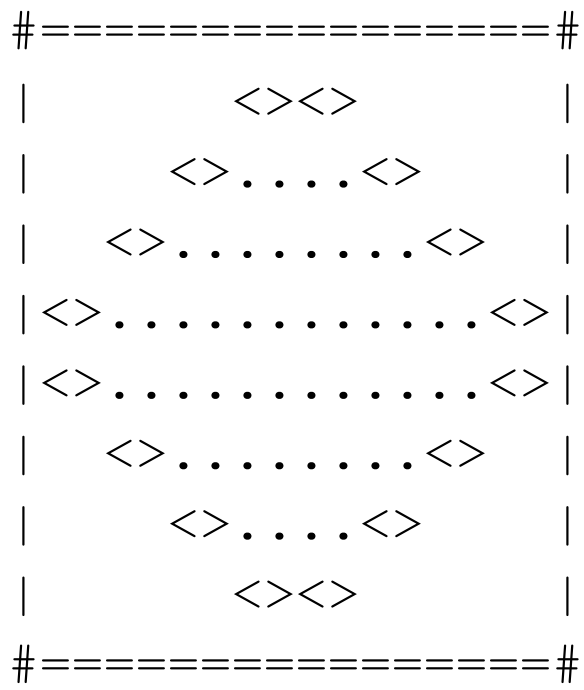
- A table for the top half:
  - Compute spaces and dots expressions from line number

line	spaces	line * -2 + 8	dots	4 * line - 4
1	6	6	0	0
2	4	4	4	4
3	2	2	8	8
4	0	0	12	12



3. Writing the code

- Useful questions about the top half:
  - Number of (nested) loops per line?



Solution for top_half()

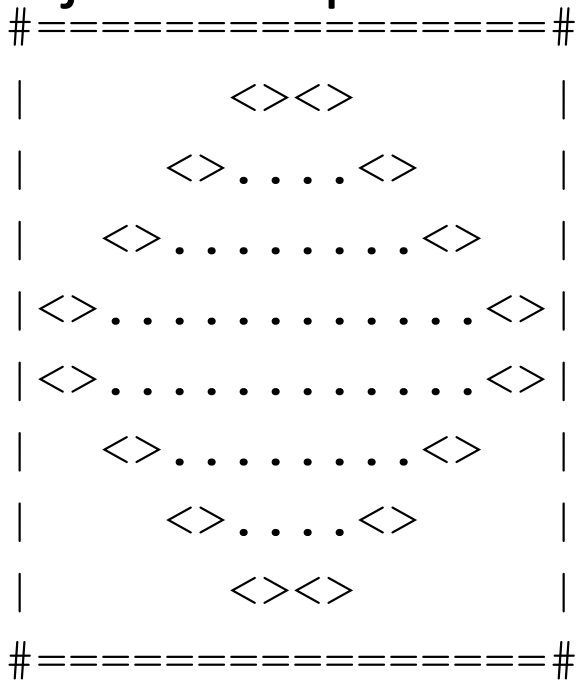
```
# Prints the expanding pattern of <> for the top half of the figure.
def top_half():
    for line in range(1, 5):
        print("|", end="")

        for space in range(1, line * -2 + 9):
            print(" ", end="")


        print("<>", end="")

        for dot in range(1, line * 4 - 3):
            print(".", end="")


        print("<>", end="")

        for space in range(1, line * -2 + 9):
            print(" ", end="")


        print("|")
```

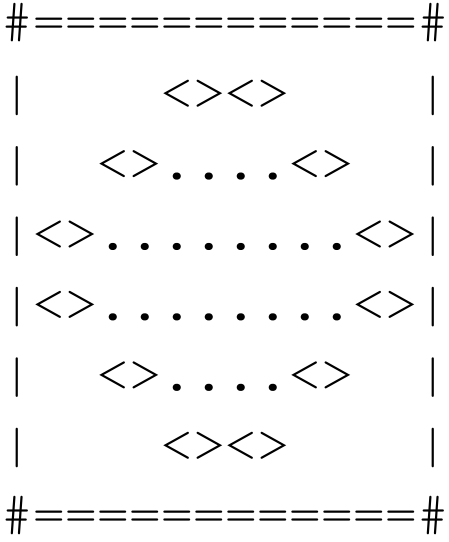
Class constants and scope

Scaling the mirror

- Let's modify our Mirror program so that it can scale.
  - The current mirror (left) is at size 4; the right is at size 3.

- We'd like to structure the code so we can scale the figure by changing the code in just one place.





Constants

- **constant**: A fixed value visible to the whole program.
  - value should only be set only at declaration; shouldn't be reassigned

- Syntax:
  - Just like declaring a normal variable:

    name = **value**

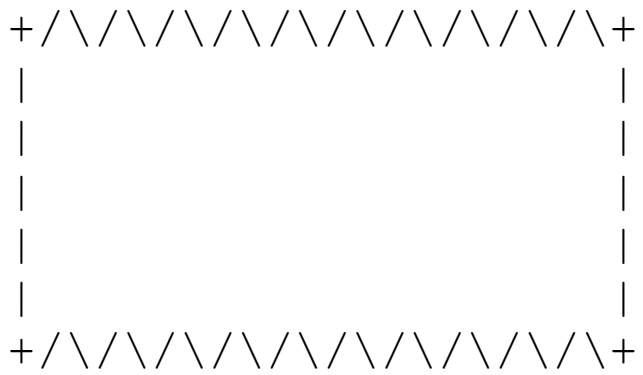
  - name is usually in ALL_UPPER_CASE

  - Examples:

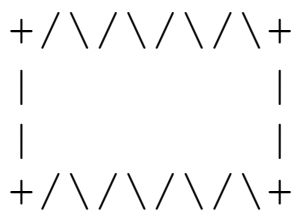
```
DAYS_IN_WEEK = 7
INTEREST_RATE = 3.5
SSN = 658234569
```

Constants and figures

- Consider the task of drawing the following scalable figure:



Multiples of 5 occur many times



The same figure at size 2

Repetitive figure code

```
def main():
    draw_line()
    draw_body()
    draw_line()

def draw_line():
    print("+", end="")
    for i in range(1, 11):
        print("/\\", end="")

    print("+")

def draw_body():
    for line in range(1, 6):
        print("|", end="")
        for spaces in range(1, 21):
            print(" ", end="")

        print("|")
```

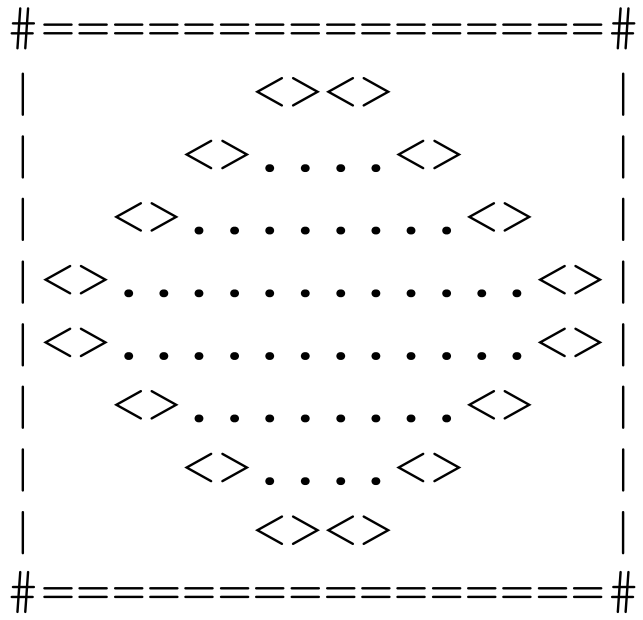
Adding a constant

```
HEIGHT = 5
def main():
    draw_line()
    draw_body()
    draw_line()

def draw_line():
    print("+", end="")
    for i in range(1, HEIGHT * 2 + 1):
        print("/\\", end="")

    print("+")

def draw_body():
    for line in range(1, HEIGHT + 1):
        print("|", end="")
        for spaces in range(1, HEIGHT * 4 + 1):
            print(" ", end="")

        print("|")
```

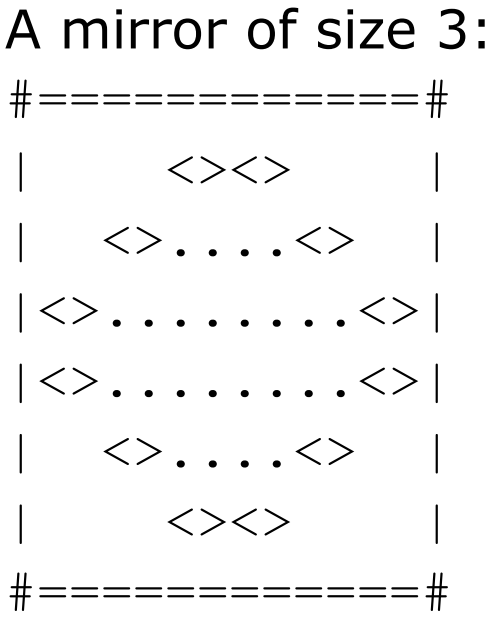
Complex figure w/ constant

- Modify the Mirror code to be resizable using a constant.

A mirror of size 4:



A mirror of size 3:



Loop tables and constant

- Let's modify our loop table to use `SIZE`
  - This can change the amount added in the loop expression

SIZE	line	spaces		dots	
4	1,2,3,4	6,4,2,0		0,4,8,12	
3	1,2,3	4,2,0		0,4,8	

```
#================#
|       <><>     |
|     <>....<>   |
|   <>........<> |
|<>............<>|
|<>............<>|
|   <>........<> |
|     <>....<>   |
|       <><>     |
#================#
```

```
#============#
|     <><>   |
|   <>....<> |
|<>........<>|
|<>........<>|
|   <>....<> |
|     <><>   |
#============#
```

Partial solution

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

        print("<>", end="")
        for dot in range(1, line * 4 - 3):
            print(".", end="")

        print("<>", end="")
        for space in range(1, line * -2 + (2*SIZE) + 1):
            print(" ", end="")

        print("|")
```

Observations about constant

- The constant can change the "intercept" in an expression.
  - Usually the "slope" is unchanged.

```
SIZE = 4;

for space in range(1, line * -2 + (2 * SIZE)):
    print(" ", end="")
```

- It doesn't replace *every* occurrence of the original value.

```
for dot in range(1, line * 4 - 4 + 1):
    print(".", end="")
```

Promoting reuse

- Programmers build increasingly complex applications
  - Enabled by existing building blocks, e.g. functions
- The more general a building block, the easier to reuse
- **Abstraction**: focusing on essential properties rather than implementation details
- Algebra is all about abstraction
  - Functions solve an entire class of similar problems

Figures with repetition

- Consider the task of printing the following lines/boxes:

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

*******

**************************************

**********
*        *
**********

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

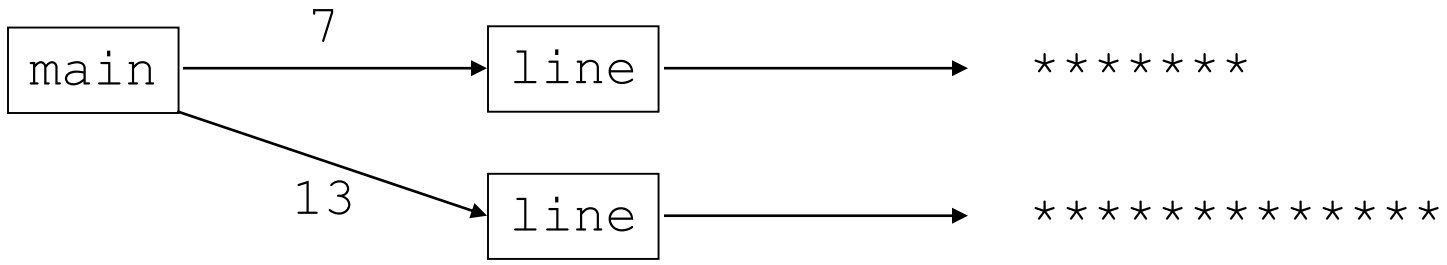
A solution with repetition

```
def main():
    line_of_13()
    line_of_7()
    line_of_35()
    box10x3()
    box5x4()

def line_of_13():
    for i in range(1, 14):
        print("*", end="")
    print()

def line_of_7():
    for i in range(1, 8):
        print("*", end="")
    print()

def line_of_35():
    for i in range(1, 36):
        print("*", end="")
    print()

...
```

- This code has repetition.
- Would variables help? Would constants help?
- What is a better solution?
  - `line` - A function to draw a line of any number of stars.
  - `box` - A function to draw a box of any size.

Parameterization

- **parameter**: A value passed to a function by its caller.
  - Instead of `line_of_7`, `line_of_13`, write `line` to draw any length.
    - When *declaring* the function, we will state that it requires a parameter for the number of stars.
    - When *calling* the function, we will specify how many stars to draw.



Declaring a parameter

Stating that a function requires a parameter in order to run

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

- Example:

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

  - When `say_password` is called, the caller must specify the code to print.

Passing a parameter

Calling a function and specifying values for its parameters

<name> (***<expression>***)

- Example:

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

Output:

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

Parameters and loops

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

```
chant(3)
```

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

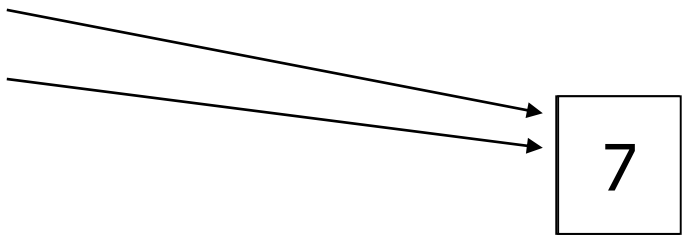
Output:

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

How parameters are passed

- When the function is called:
  - The value is stored into the parameter variable.
  - The function's code executes using that value.

```
chant(3)
chant(7)
```




```
def chant(times):
    for i in range(0, times):
        print("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
```

- Exercise: Change the `stars` program to use a parameterized function for drawing lines of stars.

Stars solution

```
# Prints several lines of stars.
# Uses a parameterized function to remove repetition.
def main():
    line(13)
    line(7)
    line(35)

# Prints the given number of stars plus a line break.
def line(count):
    for i in range(0, count):
        print("*", end="")
    print()
```

Multiple parameters

- A function can accept multiple parameters. (separate by ,)
  - When calling it, 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()
```

Output:

```
444444444
171717171717

00000000
```

- Modify the `stars` program to draw boxes with parameters.

Stars solution

```
# Prints several lines and boxes made of stars.
# Third version with multiple parameterized methods.

def main():
    line(13)
    line(7)
    line(35)
    print()
    box(10, 3)
    box(5, 4)
    box(20, 7)

# Prints the given number of
#stars plus a line break.
def line(count):
    for i in range(0, count):
        print("*", end="")
    print()
```

```
# Prints a box of stars of the given size.
def box(width, height):
    line(width)
    for line in range(0, height - 2):
        print("*", end="")
        for space in range(0, width - 2):
            print(" ", end="")
        print("*")
    line(width)
```

Strings as parameters

```
say_hello("Allison")

teacher = "Bictolia"
say_hello(teacher)

def say_hello(name):
    print("Welcome, " + name)
```

Output:

```
Welcome, Allison
Welcome, Bictolia
```

- Modify the `stars` program to use string parameters. Use a function named `repeat` that prints a string many times.

Stars solution

```
# Prints several lines and boxes made of stars.
# Fourth version with String parameters.

def main():
    line(13)
    line(7)
    line(35)
    print()
    box(10, 3)
    box(5, 4)
    box(20, 7)

# Prints the given number of
# stars plus a line break.
def line(count):
    repeat("*", count)
    print()
```

```
# Prints a box of stars of the given size.
def box(width, height):
    line(width)

    for line in range(height - 2):
        print("*", end="")
        repeat(" ", width - 2)
        print("*")
    line(width)

# Prints the given String the given
# number of times.
def repeat(s, times):
    for i in range(0, times):
        print(s, end="")
```

Value semantics

- **value semantics**: When `numbers` and `strings` are passed as parameters, their values are copied.
  - Modifying the parameter will not affect the variable passed in.

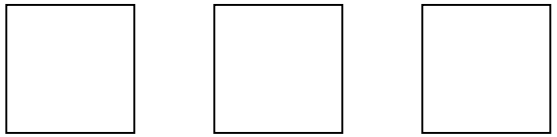
```
def strange(x):
    x = x + 1
    print("1. x = " + x)


x = 23
strange(x)
print("2. x = " + x)
...
```

Output:

```
1. x = 24
2. x = 23
```

A "Parameter Mystery" problem

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

    mystery(z, y, x)

    mystery(y, x, z)
```



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