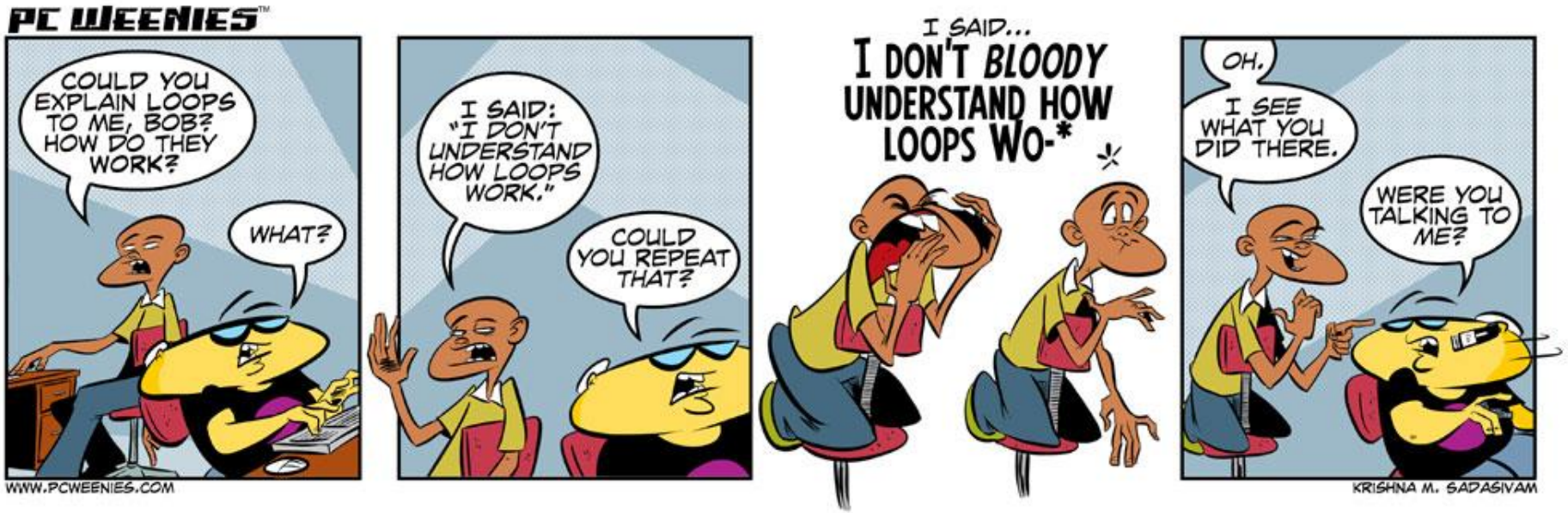
CS 115, Autumn 2021

Lecture 21: `for` loops; parameters



Thanks to Marty Stepp and Stuart Reges for parts of these slides

Mapping loops to numbers

```
for count in range(1, 6):
    print(...)
```

- What statement in the body would cause the loop to print:

  `4 7 10 13 16`

```
for count in range(1, 6):
    print(str(3 * count + 1) + " ")
```

Loop tables

```
for count in range(1, 6):
        print(…)
```

- What statement in the body would cause the loop to print:
  `2 7 12 17 22`
- To see patterns, make a table of `count` and the numbers.
  - Each time `count` goes up by 1, the number should go up by 5.
  - But `count * 5` is too great by 3, so we subtract 3.

`count`	number to print	`5 * count`	`5 * count - 3`
1	2	5	2
2	7	10	7
3	12	15	12
4	17	20	17
5	22	25	22

Loop tables question

- What statement in the body would cause the loop to print:

```
17 13 9 5 1
```

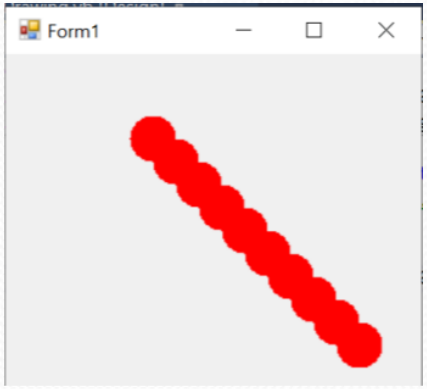
- Let's create the loop table together.
  - Each time `count` goes up 1, the number printed should ...
  - But this multiple is off by a margin of ...

`count`	number to print	`-4 * count`	`-4 * count + 21`
1	17	-4	17
2	13	-8	13
3	9	-12	9
4	5	-16	5
5	1	-20	1

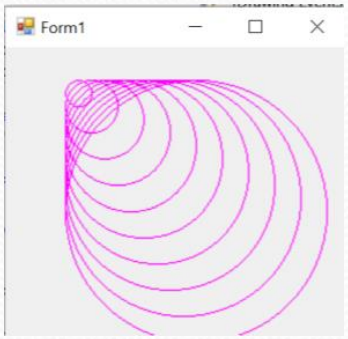
Drawing with loops

- The *x*, *y*, *w*, *h* expression can contain the loop counter, `i`.

```
for i in range(1, 11):
    panel.fill_oval(50 + 15 * i, 5 + 15 * i, 30, 30, "red")
```




```
for i in range(1, 11):
    panel.draw_oval(30, 5, 20 * i, 20 * i, "magenta")
```

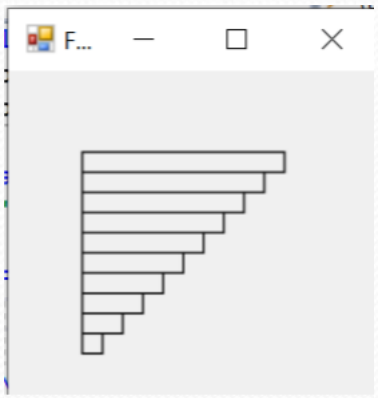


Loops that begin at 0

- Beginning a loop at 0 and using one less as the upper bound can make coordinates easier to compute.

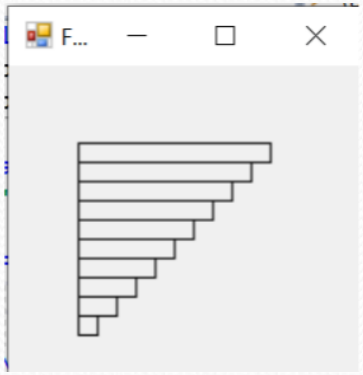
- Example:
  - Draw ten stacked rectangles starting at (20, 20), height 10, width starting at 100 and decreasing by 10 each time:

```
for i in range(0, 10):
    panel.draw_rect(20, 20 + 10 * i, 100 - 10 * i, 10)
```



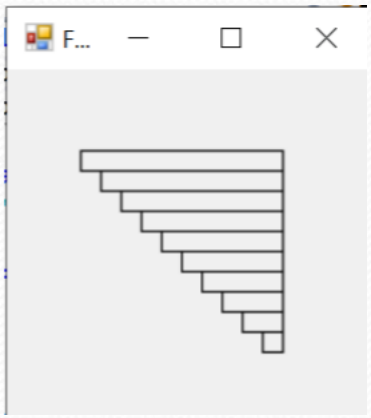
Drawing w/ loops questions

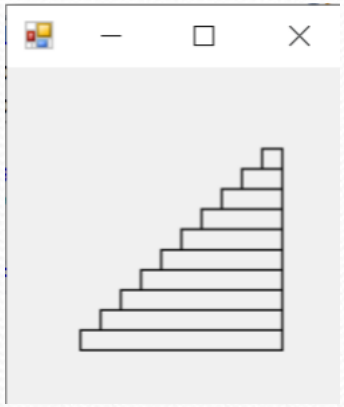
- Code from previous slide:



```
for i in range(0, 10):
    panel.draw_rect(20, 20 + 10 * i, 100 - 10 * i, 10)
```

- Write variations of the above program that draw the figures at right as output.

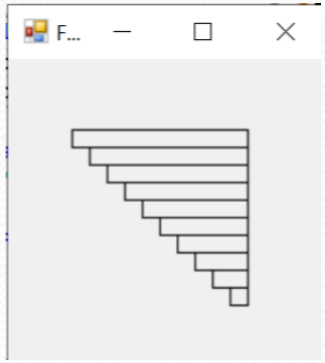




Drawing w/ loops answers

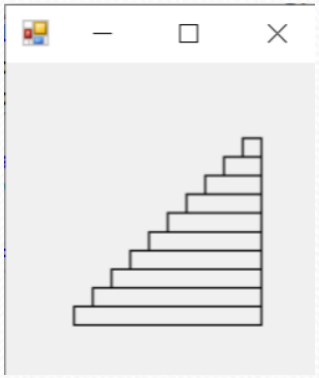
- Solution #1:

```
for i in range(0, 9):
    panel.draw_rect(20 + 10 * i, 20 + 10 * i,100 - 10 * i, 10)
```



- Solution #2:

```
for i in range(0, 9):
    panel.draw_rect(110 - 10 * i, 20 + 10 * i, 10 + 10 * i, 10)
```



Organizing our code

- Our programs have become really long! This is a problem because:
  - They are hard to read
  - It is hard to find bugs in them
  - Often the same code is repeated
    - We can eliminate part of this with loops
      - Can we eliminate all of it?
- Python contains a lot of built-in commands. Can we make our code into new commands?

Creating commands

- Python commands like `print()` are just functions
- Recall: a ***function*** is a way to group statements together and give them a name.
- It would have been nice to put each animal in its own function in the ***Stamps*** project.
  - Why couldn't we do this?

Example: `bird`

- Code to draw a bird from lab 5:

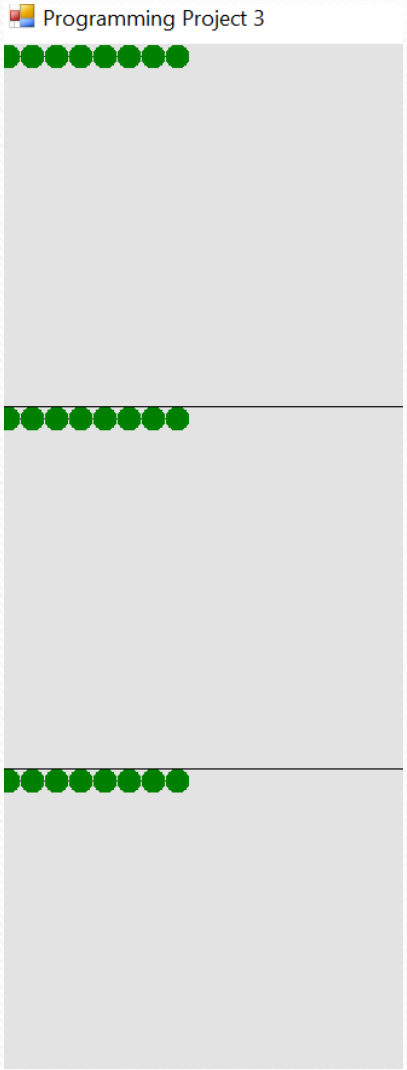
```
panel.fill_oval(x + 5, y, BIRD_WIDTH - 10, BIRD_HEIGHT, "#990099") # body
panel.fill_oval(x + 40, y + 25, 20, 40, "#770077")                 # right wing
panel.fill_oval(x, y + 25, 20, 40, "#770077")                      # left wing
panel.fill_oval(x + 20, y + 10, 5, 5, "black")                     # left eye
panel.fill_oval(x + 35, y + 10, 5, 5, "black")                     # right eye
panel.fill_rect(x + 28, y + 18, 4, 8, "#eecc00")                   # beak
```

Scope

- A variable's scope is the part of the program where Python knows it exists and allows you to use it
  - Remember: variables created inside a function only exits in that function.
    - If we try to access a variable outside of our scope Python either creates a new variable with the same name or gives us an error

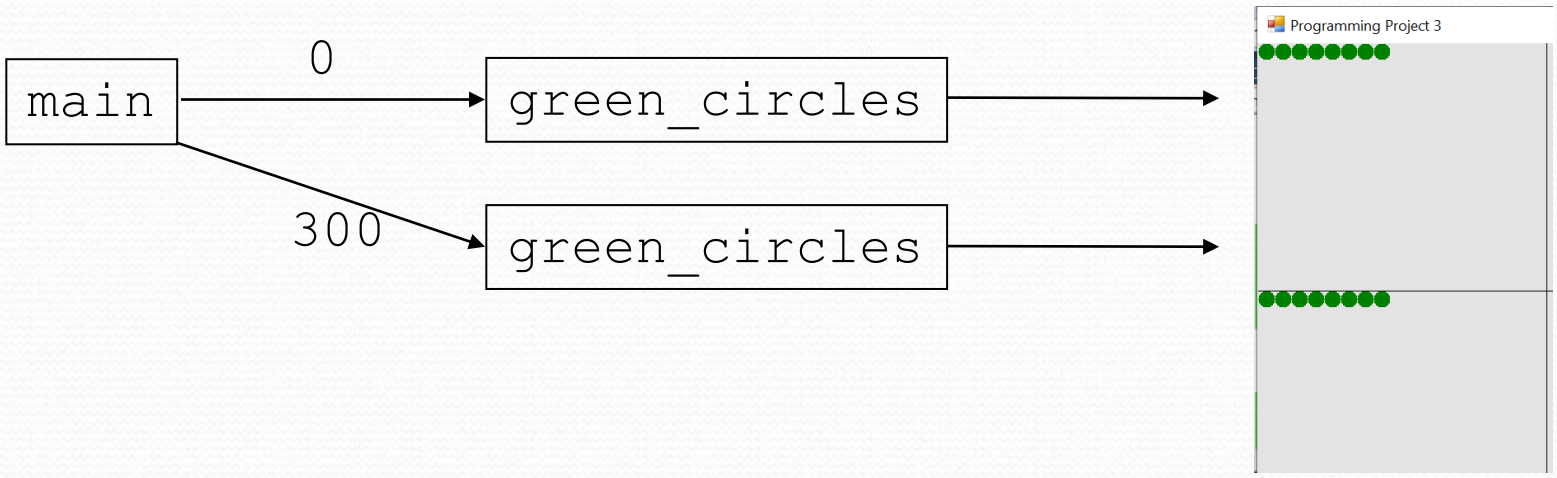
Parameters

- Parameters allow us to send information to a function
  - We can make a function that will do many similar tasks
    - For example, draw a pattern of circles at different locations
  - Sometimes also called arguments



Parameterization

- **parameter**: A value passed to a function by its caller.
  - Instead of `green_circles_top_left`, `green_circles_middle_left`, write `green_circles` to draw at any location.
    - When *declaring* the function, we will state that it requires a parameter for the y coordinate.
    - When *calling* the function, we will specify what y to draw at.



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: " + str(code))
```

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

Passing a parameter

Calling a function and specifying values for its parameters

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

- Example:

```
def main()
    say_password(42)
    say_password(12345)
```

Output:

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

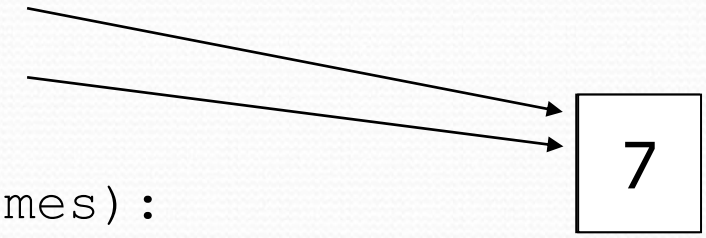
How parameters are passed

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

```
def main()
    chant(3)
    chant(7)

def chant(times):
    print("Just", times, "salads...")

main()
```



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
```

- Exercise: Change the `green_circles` program to use a parameterized function for drawing lines of green circles at different locations.

Multiple parameters

- A subroutine 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_numbers(4, 9)
    print_numbers(17, 6)
    print_numbers(8, 0)
    print_numbers(0, 8)
def print_numbers(number, count):
    for i in range(1, 6):
        print(str(number) + str(count) + " ")
    print ()
```

Output:

```
49 49 49 49 49
176 176 176 176 176
80 80 80 80 80
08 08 08 08 08
```

- Modify the `green_circles` program to draw rows at different x locations.

Green circles solution

```
def main():
    panel = drawing_panel(700, 700)
    green_circles(panel, 0, 0)
    green_circles(panel, 20, 300)
    green_circles(panel, 150, 600)

def green_circles(panel, x, y):
    for i in range(0 7):
        panel.fill_oval(x + i * 20, y, 20, 20, "green")

main()
```

Parameters and loops

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

```
def main():
    chant(3)

def chant(times)
    for i in range(1, times + 1)
        print("Just a salad...")
```

Output:

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

Value semantics

- **value semantics**: When value types (`int`, `double`) 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)

def main()
    x = 23
    strange(x)
    print("2. x =", x)
    ...

main()
```

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))


main()
```

Other types as parameters

```
def main():
    say_hello("Merlin")
    cat = "Sir Purrcival"
    say_hello(cat)

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

Output:

```
Welcome, Merlin
Welcome, Sir Purrcival
```

- Modify the `green_circles` program to use color parameters.

Green circles solution

```
def main():
    panel = drawing_panel(700, 700)
    green_circles(0, 0, 7, "green")
    green_circles(20, 300, 4, "yellow")
    green_circles(150, 600, 12, "#CC5555")

def green_circles(panel, x, y, count, color):
    for i in range(0, count + 1):
        panel.fill_oval(x + i * 20, y, 20, 20, color)

main()
```