CS 115, Autumn 2021

Lecture 16: boolean logic and `while` loops

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

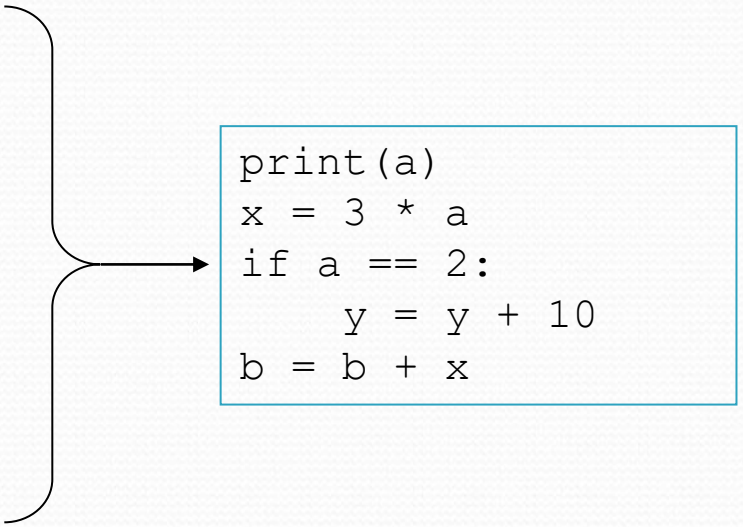
Exercise: bouncing ball

- Draw a ball on a `drawing_panel` and animate its path as it falls.
  - It should bounce when it hits the ground
  - Add a count for how many times the ball bounces
  - Can we make it fall with gravity?
    - displacement = velocity * time + ½ * acceleration * time^2

Factoring `if/else` code

- **factoring**: Extracting common/redundant code.
  - Can reduce or eliminate redundancy from `if/else` code.
- Example:

```
if a == 1:
    print(a)
    x = 3
    b = b + x
elif a == 2:
    print(a)
    x = 6
    y = y + 10
    b = b + x
else:          # a == 3
    print(a)
    x = 9
    b = b + x
```



```
print(a)
x = 3 * a
if a == 2:
    y = y + 10
b = b + x
```

Type `bool`

- **`bool`**: A logical type whose values are `True` and `False`.
  - A logical ***test*** is actually a Boolean expression.
  - Like other types, it is legal to:
    - create a `bool` variable

```
minor      = age < 21
is_prof    = "Prof" in name
loves_csc = True

# allow only CS-loving students over 21
if minor or is_prof or not loves_csc:
    print("Can't enter the club!")
```

Using `bool`

- Why is type `bool` useful?
  - Can capture a complex logical test result and use it later
  - Can write a function that does a complex test and returns it
  - Makes code more readable

```
good_age    = age >= 27 and age < 39
good_height = height >= 78 and height < 84
rich        = salary >= 100000.0
if (goodAge and goodHeight) or rich:
    print("Okay, let's go out!")
else:
    print("It's not you, it's me...")
```

De Morgan's Law

- **De Morgan's Law**: Rules used to negate boolean tests.
  - Useful when you want the opposite of an existing test.

Original Expression	Negated Expression	Alternative
`a and b`	`not a or not b`	`not(a and b)`
`a or b`	`not a and not b`	`not(a or b)`

  - Example:

Original Code	Negated Code
`if x == 7 and y > 3:` `    ...`	`if x != 7 or y <= 3:` `    ...`

Repetition

- So far, repeating an action results in redundant code:

```
draw_ball()
draw_stripe()
draw_stripe()
draw_stripe()
draw_stripe()
draw_stripe()
draw_ground()
```

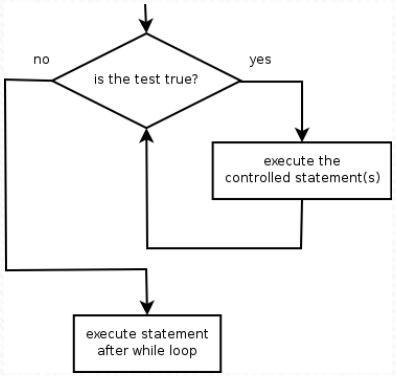
- Python's **`while` loop** statement performs a task many times.

```
draw_ball()
i = 1
while i <= 5:     # repeat 5 times
    draw_stripe()
    i = i + 1
draw_ground()
```

The `while` loop

- **`while` loop**: Repeatedly executes its body as long as a logical test is true.

```
while test:
    statement(s)
```



- Example:

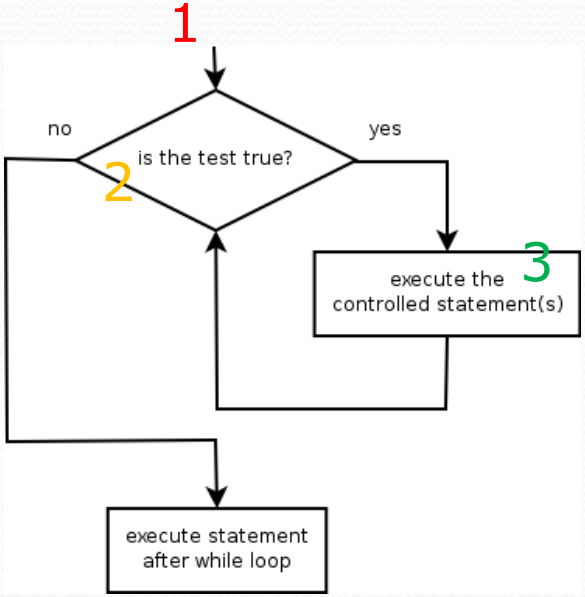
```
num = 1                                  # initialization
while num <= 200:                        # test
    print(str(num) + " ", end='')
    num = num * 2                        # update

# output:  1 2 4 8 16 32 64 128
```

Example `while` loop

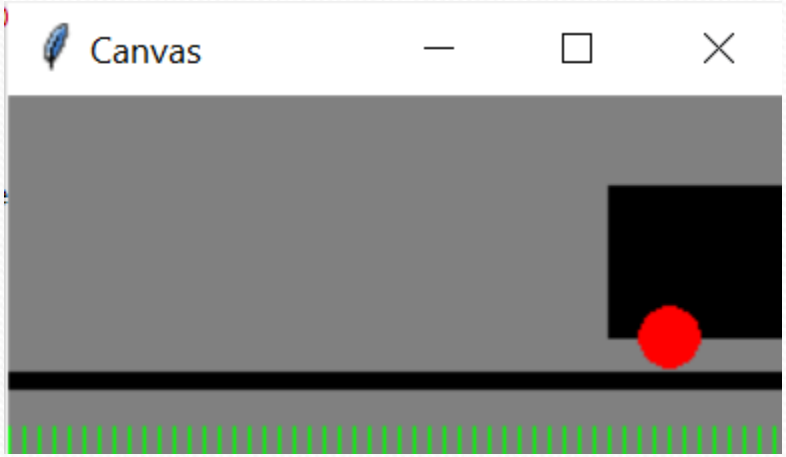
```
# finds the first factor of 91, other than 1
n = 91
factor = 2
while n % factor != 0:
    factor += 1
print("First factor is", factor)
# output:  First factor is 7
```

1 2 3 4



Exercise

- Change the single green line at the bottom of the car program to be a line of separate blades of grass



Exercise

- Alter the car animation to make it slowly accelerate.
  - Can we do this with `panel.animate(`**ms**, **function**`)`?
    - No – we set the speed once.
  - Instead, use `panel.sleep(`**ms**`)` to make the animation pause for the passed in time.
    - How can we make this repeat?
      - A `while` loop!