CS 115, Autumn 2021

Lecture 12: `global`; `if` / `else`

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

Stroke Weight

- Set the thickness of lines and borders with:

```
panel.set_stroke(5)
```

  - The default stroke weight is 1
  - Once you set the stroke weight, it stays at that weight until you reset it

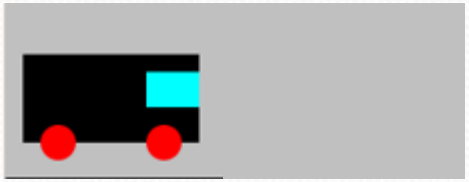
Constants

- When we have a constant value that our program depends on and we might want to change it later it is helpful to store this in a variable.
  - Make this a special variable that you guarantee will stay the same throughout an entire run of your program
  - Place it in an easy to find spot so the user can find it to change it between runs – convention: at the top of your file
  - Naming convention: ALL_CAPS
  - Example

```
PANEL_WIDTH = 500
PANEL_HEIGHT = PANEL_WIDTH // 2
```

Moving in a pattern

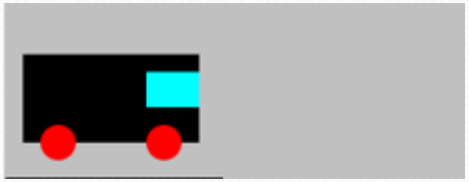
- How can we move a shape in a specific pattern?
  - Example: make the car appear to drive across the panel
  - We will need to keep track of the x location of the car and make it increase a little each time we draw the car
    - How can we keep track of this?



```
def repeat():
    x = 10
    y = 30
    panel.fill_rect(x, y, 100, 50, "black")
    panel.fill_oval(x + 10, y + 40, 20, 20, "red")
    panel.fill_oval(x + 70, y + 40, 20, 20, "red")
    panel.fill_rect(x + 70, y + 10, 30, 20, "cyan")

panel = drawing_panel(260, 100, "gray")
panel.animate(50, repeat)
```

Can we use variables?

- Option 1: declare some variables inside our animation function
  - problem: how do we make the x, y position change?
  - possible solution: set the variables to themselves plus something
    - problem: what are x and y the first time?
      - this doesn't work

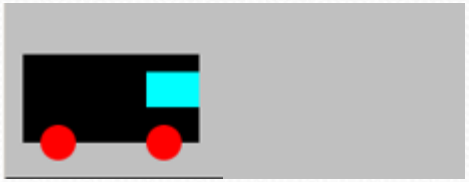


```
def repeat():
    x = x + 10
    y = y + 10
    panel.fill_rect(x, y, 100, 50, "black")
    panel.fill_oval(x + 10, y + 40, 20, 20, "red")
    panel.fill_oval(x + 70, y + 40, 20, 20, "red")
    panel.fill_rect(x + 70, y + 10, 30, 20, "cyan")

panel = drawing_panel(260, 100, "gray")
panel.animate(50, repeat)
```

Can we use variables?

- Option 2: declare some variables **outside** our animation function
  - increase them **inside** the animation function
  - problem: our code won't run! Why not?
    - by default we can only change variables created inside a function when we are in a function

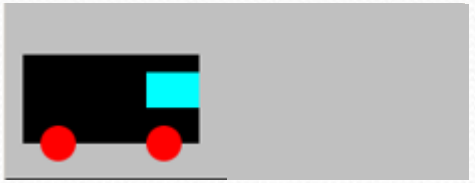
```
x = 10
y = 30

def repeat():
    x = x + 10
    y = y + 10
    panel.fill_rect(x, y, 100, 50, "black")
    panel.fill_oval(x + 10, y + 40, 20, 20, "red")
    panel.fill_oval(x + 70, y + 40, 20, 20, "red")
    panel.fill_rect(x + 70, y + 10, 30, 20, "cyan")

panel = drawing_panel(260, 100, "gray")
panel.animate(50, repeat)
```



global

- The **`global`** keyword allows us to change a variable from **outside** of a function when we are **inside** a function
  - create the variable outside of the function
  - before you assign the variable inside the function include a line with: `global` **variable_name**
  - now use this variable just as you would any other variable

```
x = 10
y = 30

def repeat():
    global x
    global y
    x = x + 10
    y = y + 10
    panel.fill_rect(x, y, 100, 50, "black")
    panel.fill_oval(x + 10, y + 40, 20, 20, "red")
    panel.fill_oval(x + 70, y + 40, 20, 20, "red")
    panel.fill_rect(x + 70, y + 10, 30, 20, "cyan")

panel = drawing_panel(260, 100, "gray")
panel.animate(50, repeat)
```



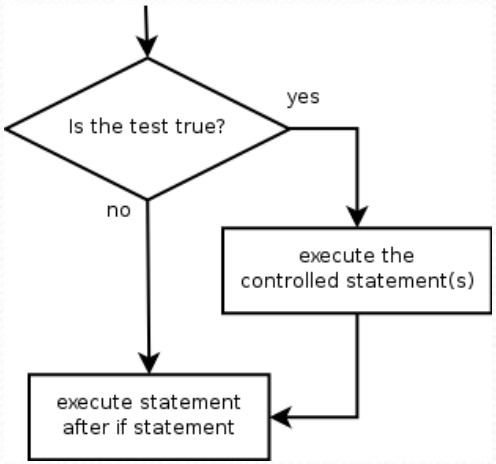
What if we want to wrap around the screen?

- In many games and animations when something reaches the edge of the screen, it appears on the other side.
  - What do we need to change to make this happen?
  - How can we only make this change when we reach the edge of the screen?
    - If we do this on every move we will appear to stay on the left side and not move

The `if` statement

Executes a block of statements only if a test is true

```
if test:
    statement
    ...
    statement
```



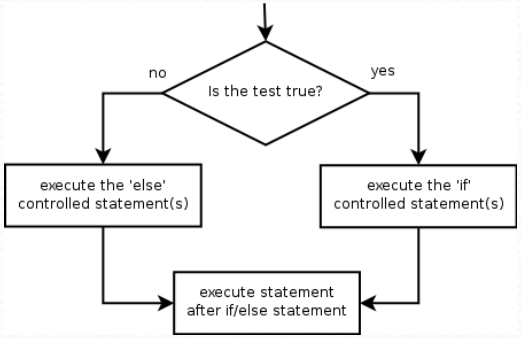
- Example:

```
gpa = float(input("gpa? "))
if gpa >= 2.0:
    print("Application accepted.")
```

The `if/else` statement

Executes one block if a test is true, another if false

```
if test:
    statement(s)
else:
    statement(s)
```



- Example:

```
gpa = float(input("gpa? "))
if gpa >= 2.0:
    print("Welcome to Mars University!")
else:
    print("Application denied.")
```

Relational expressions

- `if` statements use logical tests.

```
if i <= 10: ...
```

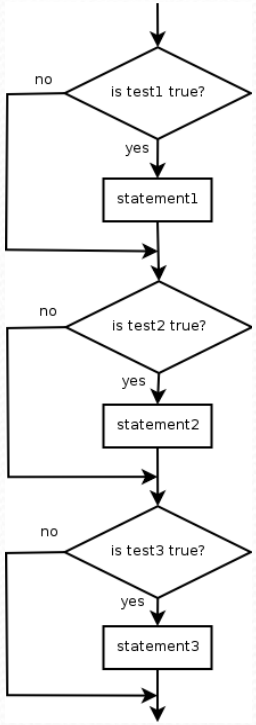
  - These are `boolean` expressions
- Tests use *relational operators*:

Operator	Meaning	Example	Value
==	equals	1 + 1 == 2	True
!=	does not equal	3.2 != 2.5	True
<	less than	10 < 5	False
>	greater than	10 > 5	True
<=	less than or equal to	126 <= 100	False
>=	greater than or equal to	5.0 >= 5.0	True

Misuse of `if`

- What's wrong with the following code?

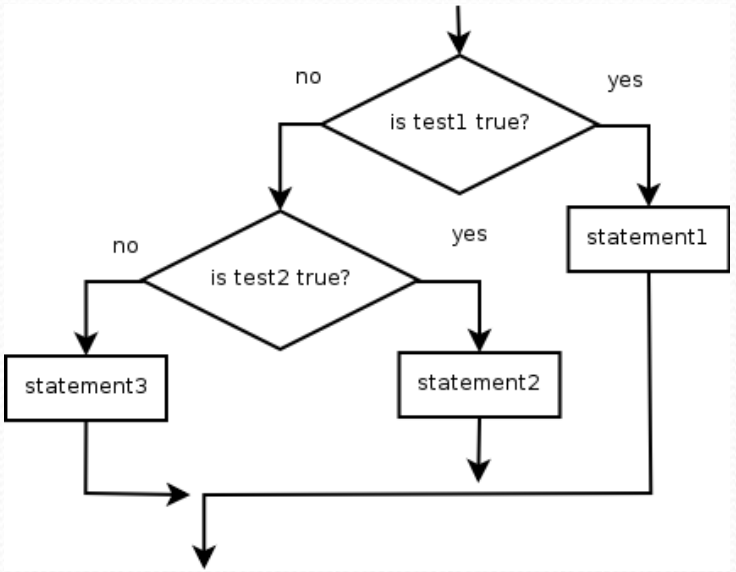
```
percent = float(input("What percentage did you earn? "))

if percent >= 90:
    print("You got an A!")

if percent >= 80:
    print("You got a B!")

if percent >= 70:
    print("You got a C!")

if percent >= 60:
    print("You got a D!")

if percent < 60:
    print("You got an F!")

...
```



Nested `if/else`

Chooses between outcomes using many tests

```
if test:
    statement(s)
elif test:
    statement(s)
else:
    statement(s)
```



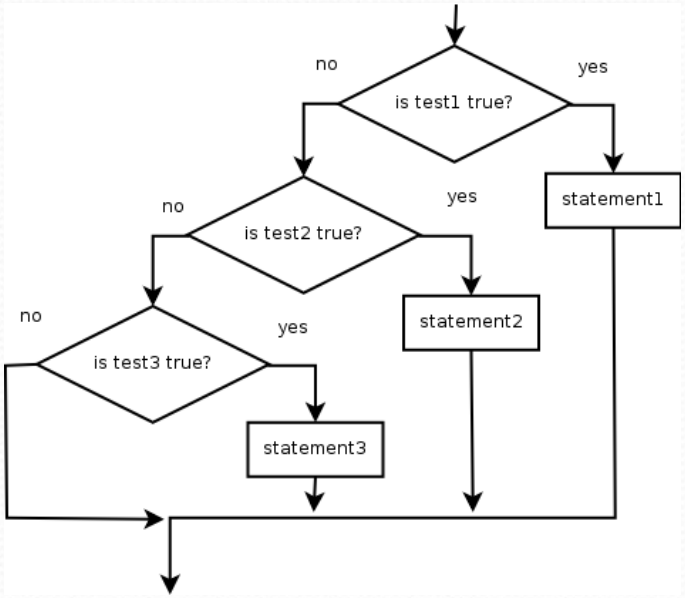
- Example:

```
if x > 0:
    print("Positive")
elif x < 0:
    print("Negative")
else:
    print("Zero")
```

Nested `if/elif/elif`

- If it ends with `else`, exactly one path must be taken.
- If it ends with `if`, the code might not execute any path.

```
if test:
    statement(s)
elif test:
    statement(s)
elif test:
    statement(s)
```



- Example:

```
if place == 1:
    print("Gold medal!")
elif place == 2:
    print("Silver medal!")
elif place == 3:
    print("Bronze medal.")
```

Nested `if` structures

- exactly 1 path *(mutually exclusive)*

```
if test:
    statement(s)
elif test:
    statement(s)
else:
    statement(s)
```

- 0 or 1 path *(mutually exclusive)*

```
if test:
    statement(s)
elif test:
    statement(s)
elif test:
    statement(s)
```

- 0, 1, or many paths *(independent tests; not exclusive)*

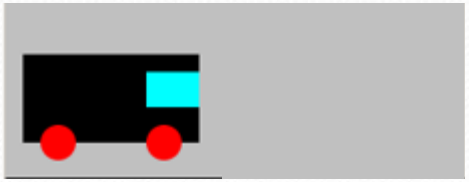
```
if test:
    statement(s)

if test:
    statement(s)

if test:
    statement(s)
```

Which nested `if/else`?

- **(1) if/if/if (2) nested if/else (3) nested if/elif/elif**
  - Whether a user is lower, middle, or upper-class based on income.
    - **(2)** nested `if / elif / else`
  - Whether you made the dean's list (GPA ≥ 3.8) or honor roll (3.5-3.8).
    - **(3)** nested `if / elif`
  - Whether a number is divisible by 2, 3, and/or 5.
    - **(1)** sequential `if / if / if`
  - Computing a grade of A, B, C, D, or F based on a percentage.
    - **(2)** nested `if / elif / elif / elif / else`

Exercise: driving

- Make the car reappear at the left side of the screen after it reaches the left side.



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