CIS 1100

Conditionals (Lecture)

Python

Fall 2024

University of Pennsylvania

Recap: Conditions as Boolean Expressions

Boolean expressions evaluate to `bool` values, i.e. either `True` or `False`.

```
3 < 4 and 9 == (81 / 9)                  # always True
not True and True or False and not False # always False
```

We are also able to write boolean expressions that contain variables.

```
x % 3 == 2 and x > 5                     # not always True or False!
```

This expression's value changes based on the value of `x`!

Can you think of a value of x *that would cause the expression to evaluate to* `True`*? What about* `False`*?*

The Boolean Expression Toolkit

Relational Operators:

Operator/method	Input Types	Description
`<` / `<=`	`int`, `float`, `str`	less than / less than or equal to
`>` / `>=`	`int`, `float`, `str`	greater than / greater than or equal to
`==` / `!=`	`int`, `float`, `str`	equal to / not equal to

The Boolean Expression Toolkit

Logical Operators:

Operator/method	Input Types	Description
`and`	`bool`	evaluates to `True` only if both inputs are `True`
`or`	`bool`	evaluates to `True` as long as at least one input is `True`
`not`	`bool`	negates a single `bool` value to its opposite

Activity: Under Pressure

I'm writing a program to monitor valve pressure in a chemical plant. I want to define *safe conditions* as those where the pressure is between `0.5` and `3.5`. Which is a boolean expression that is `True` only when conditions are safe? (M1)

- (A) `0.5 < pressure and 3.5 < pressure`
- (B) `0.5 < pressure and 3.5 > pressure`
- (C) `0.5 < pressure or 3.5 > pressure`
- (D) `0.5 > pressure or 3.5 < pressure`
- (E) `0.5 > pressure and 3.5 < pressure`

Activity: Satisfaction

Find a value for `s` such that this expression is `True`, or write "None" if there are none: (S7)

```
len(s) > 5 and len(s) % 2 == 0 and s.find("watch") == 5
```

Find a value for `x` such that this expression is `True`, or write "None" if there are none: (S8)

```
(3 < x < 8 or x % 2 == 0) and (x // 10 == 0 or x % 2 != 0)
```

Find a value for `x` such that this expression is `True`, or write "None" if there are none: (S9)

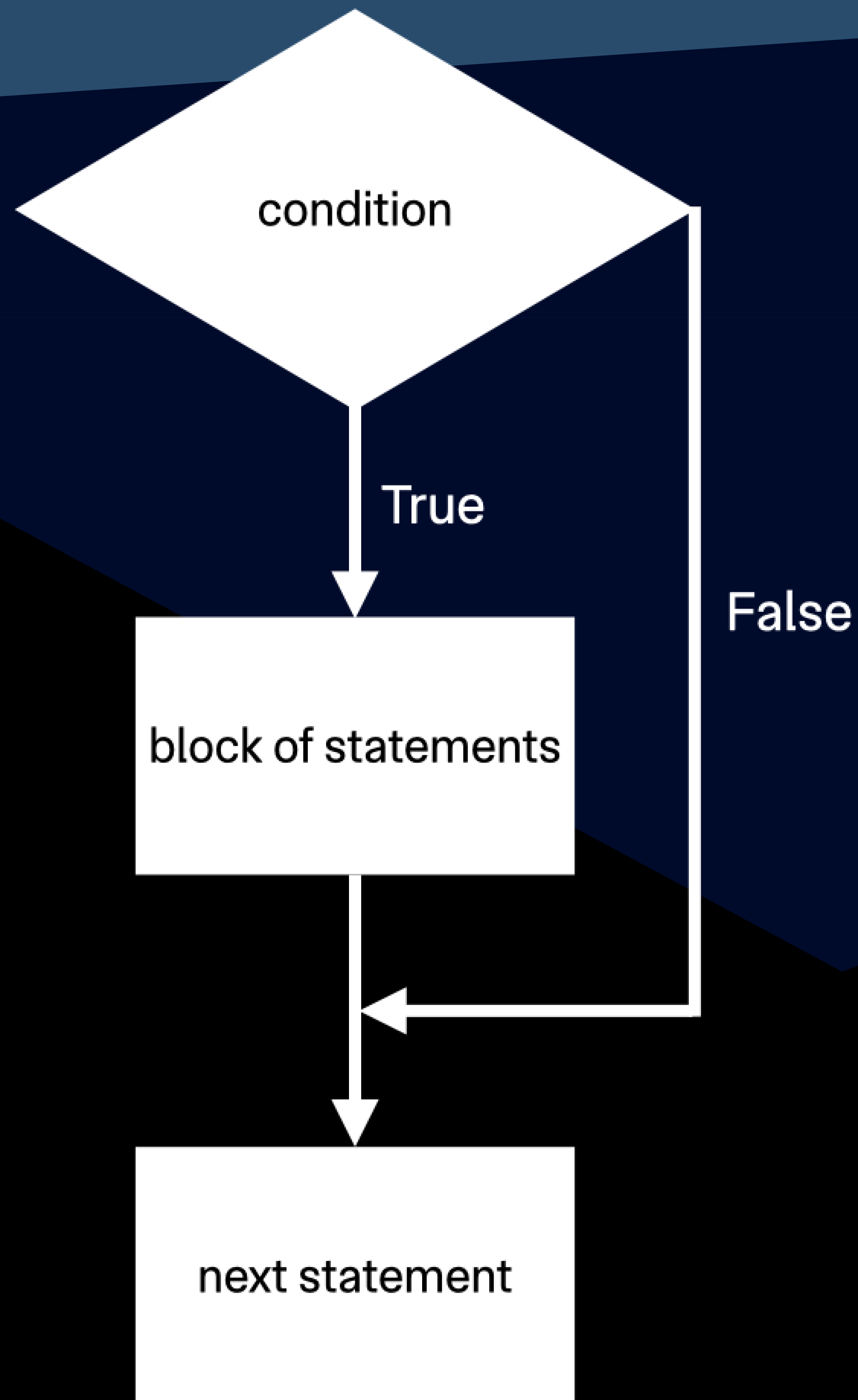
```
(3 < x < 8 and x % 2 == 0) and (x // 10 == 0 and x % 2 != 0)
```

Recap: `if`

> *"`if` music be the food of love, play on."* — William Shakespeare

The `if` statement allows us to specify a portions of our program that should be run **only in the case that** a certain condition is met.

```
if my_boolean_expression:
        statement_one
        statement_two
        ...
        statement_last
```



Recap: Control Flow & `if`

- Test the condition...
  - **if** it is `True`, execute the block of statements
  - otherwise, proceed to the next statement.

Activity:

```python
import penndraw as pd
r = 0.1
if r == 0.1:
    pd.set_pen_color(pd.RED)
if r > 0.05:
    pd.set_pen_color(pd.GREEN)
if r < 0.5:
    pd.filled_circle(0.5, 0.5, r)
    pd.set_pen_color(pd.BLACK)
```

What color is the circle that gets drawn? (M2)

(A) Red

(B) Green

(C) Black

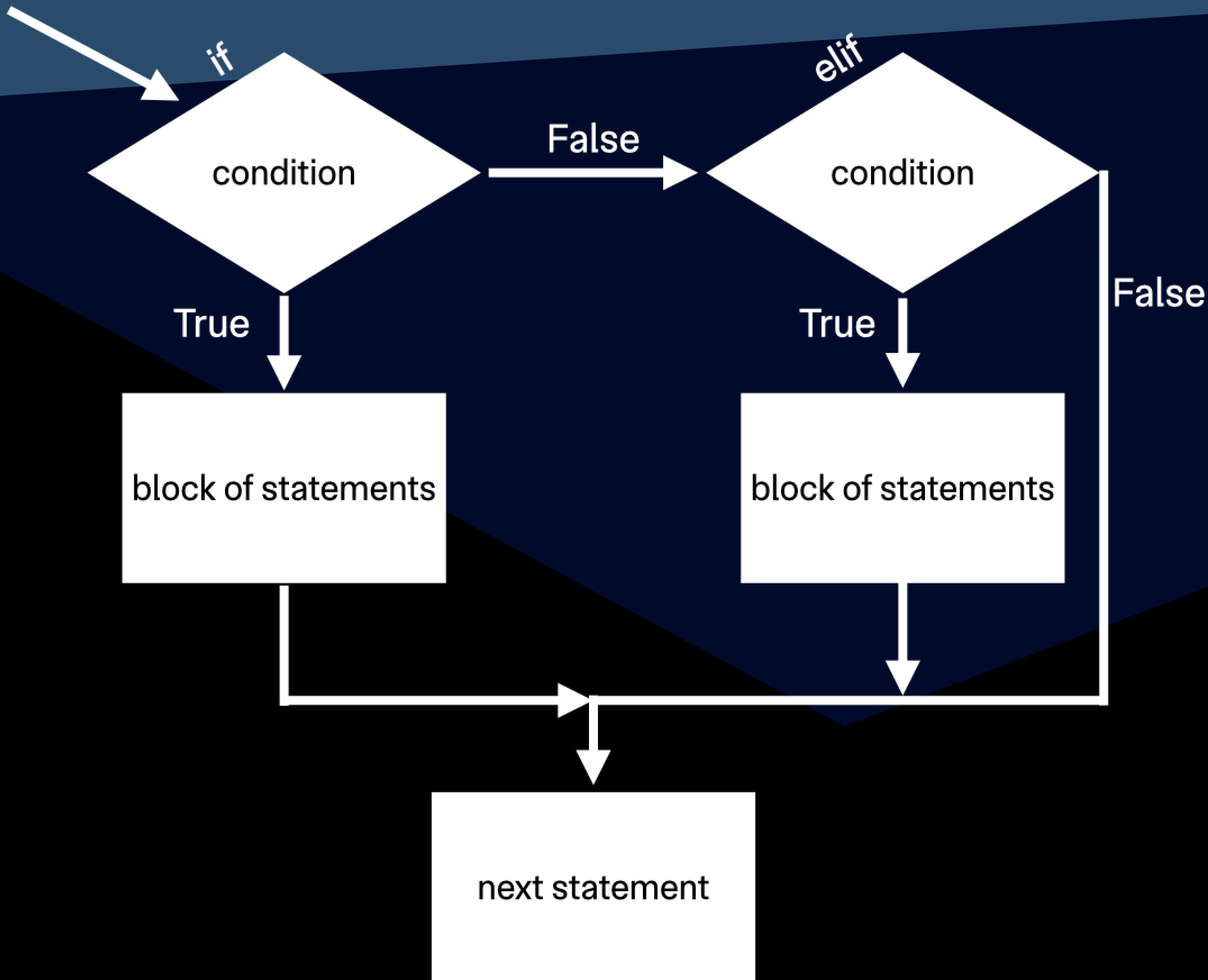
(D) There is no circle drawn

Recap: `elif`

`elif` allows you to specify an *alternative condition* that is be tested *only when all previous conditions were* `False`.

The `elif` syntax:

```
if first_boolean_expression:
        statement_one
        statement_two
        ...
        statement_last
elif alternative_boolean_expression:
        statement_a
        statement_b
        ...
        statement_z
```



Recap: `elif`

`if` and `elif` statements represent *mutually exclusive* choices: we may execute the body of one, the other, or neither, but *never both*.

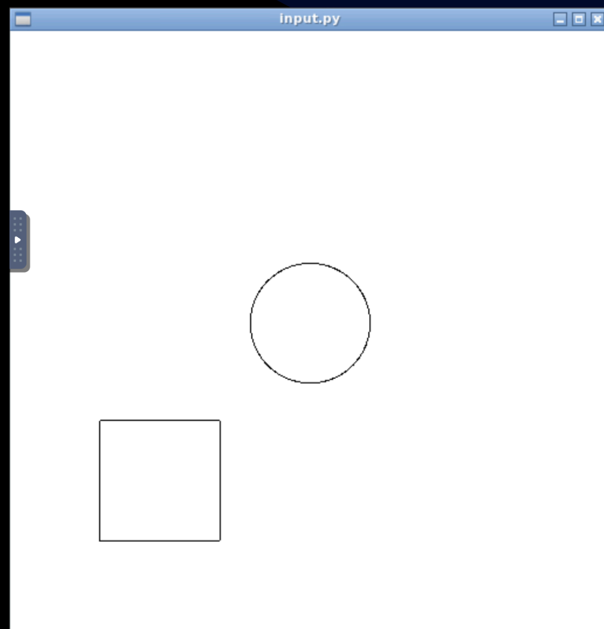
Activity: Picking Paths

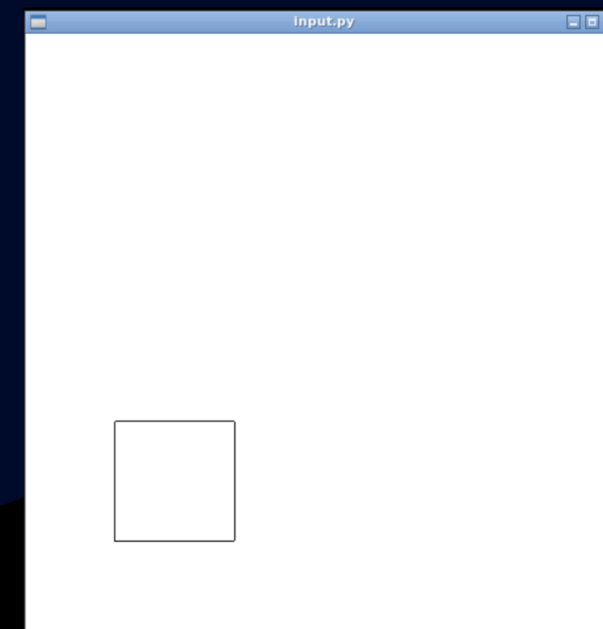
1. Draw a flow-chart representing the control flow of this program. Use diamonds for conditionals (`if`/`elif`) and put the boolean expressions inside of the diamonds. Represent blocks of code as rectangles. Write all of the lines of code that belong to a block in the rectangle. (C12)

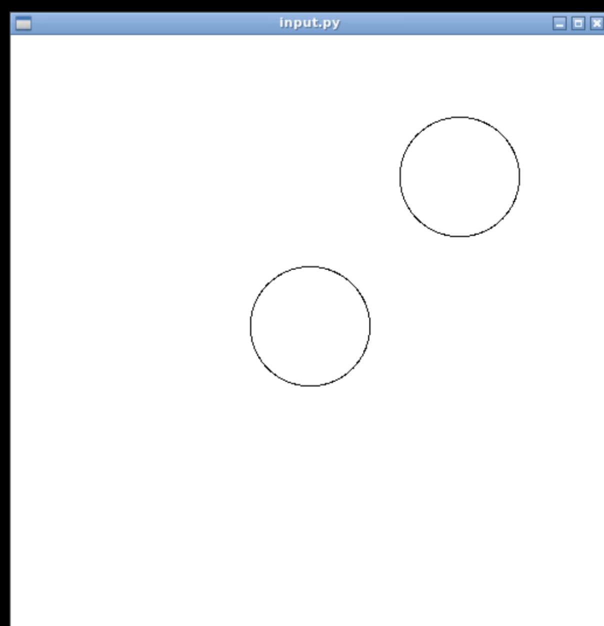
```
import penndraw as pd
code = input("TYPE THE CODE")

if len(code) == 4:
    pd.square(0.25, 0.25, 0.1)
elif code[0] == "a":
    pd.circle(0.75, 0.75, 0.1)

if code.find("a") == 0:
    pd.circle(0.5, 0.5, 0.1)
pd.run()
```

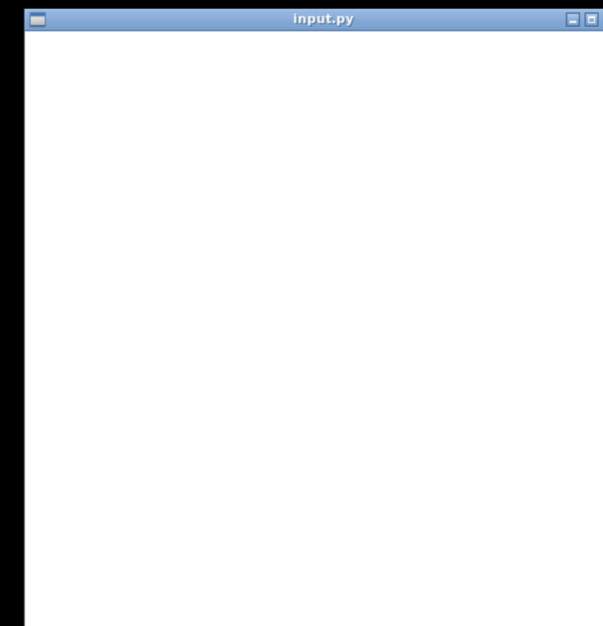
Activity: Picking Paths

Clockwise from the top-left, write a "CODE" that a person could type in to generate each output. (L13)









Recap: `else`

The `else` keyword allows us to define a body of statements that will be run *no matter what* in the case that all previous conditions were not met.

```
if first_boolean_expression:
    block_one
elif alternative_boolean_expression:
    block_two
# optionally many elif statements provided here...
else:
    block_three
```

Look: no new condition provided!

Recap: All Conditionals

Recipe for any conditional:

1. Always start with an `if`. Each `if` comes with a boolean expression to test. This expression is always tested.
2. As many `elif` statements as desired. Each comes with a boolean expression. Each expression only tested if all previous are `False`.
3. An `else` statement, or not. No boolean expression provided. Body executed if all previous expressions are `False`.

Activity: Fix it!

Both snippets below are broken for some reason. In (C14), write an explanation for what is wrong in both cases.

```
x = int(input())
if x > 12:
    print("daylight")
elif x < -10:
    print("fires")
elif x != 45:
    print("ridges")
else x % 13 == 2:
    print("green")
```

```
y = input()
if y == "bliss":
    z = "ful"
else:
    print(y + z)
```

Activity `hot_or_cold.py`

Write a program that simulates a guessing game. (The answer is always 50, but the gamer is ignorant of this fact.)

- You will need to prompt the user for input using `input()` and parse that value as a number using `int()`.
- If their guess is correct, print out "Victory".
- If the guess is within ten of the correct answer, print out "Hot 🥵".
- Otherwise, print out "Cold 🥶"

(C16)

Recap: `case` / `match`: Another Way to Choose

What to do at a traffic light, take two:

```
match traffic_light:
    case "red":
        print("Stop!")
    case "yellow":
        print("Slow down.")
    case "green":
        print("Proceed carefully.")
```

- `match` allows us to compare an expression's value to several different cases.
- Each `case` gives a value to compare to and a block of code to execute if there's a match.
- Use `|` to specify multiple options per case
- Use `_` to specify a fall-back

Activity: Tier List

A *tier list* is an assignment of letter grades to different options.

CIS 1100 TAs are really opinionated about lots of things, including different kinds of milks. The consensus is the following:

Milks	Tier
Oat	S (highest)
Cow, soy	A
Everything else	B
Almond	F (lowest; a disgrace)

Use `match` and `case` to write a program that prints the tier of a milk name. (C16)

```python
milk = input()
match milk:
    case "Oat":
        print("S")
    case "Cow" | "Soy":
        print("A")
    case "Almond":
        print("F")
    case _:
        print("B")
```

Reminders

- HW00 is due tonight at 11:59pm
  - Use one late day --> submit by 9/12 @ 11:59pm
  - Use two late days --> submit by 9/13 @ 11:59pm
- Earn late tokens by handing in effortfully completed worksheets
- Check-in due before 9/13
- HW01 Released Tomorrow Afternoon, Due 9/18
  - START EARLY!
- Sunday Review Session each Sunday 10am-noon
- Recitation continues next week