CIS 1100

Conditionals (Lecture)

Python

Fall 2024

University of Pennsylvania

Sequence Types: String

An important aspect of the string type, is that it is a sequence type. A string contains a *sequence* of characters.

Sequences have a length and indexes for individual members of the sequences.

For strings, it is a sequence of characters. Emphasis on sequences because: **The order matters**

```
s1 = "hi"
s2 = "ih"
print(s1 == s2) # False
```

Sequence Types: String

Ordering is an important aspect to string, and we use **indexes** as a way to specify positions in the string.

Consider the string

index	**0**	**1**	**2**	**3**	**4**	**5**
characters	H	e	l	l	o	!

index `0` is the first position (first character)

`len(string) - 1` is the index of the last character

Sequence Types: Basic Functionality

For any sequence, you can use

- `len(seq)` to find the length of the sequence `seq`
- `seq[i]` to access the `i`th index of the sequence `seq`

```
x = "Maracuyá"
print(x[0])          # 'M'
print(x[3])          # 'a'
print(x[len(x) - 1]) # 'á'
```

Practice:

What is the index of the letter `B` from the following string (M3)

```
x = "GY! BE"
```

- (A) `3`
- (B) `4`
- (C) `5`
- (D) `2`
- (E) `0`

Extra Practice:

Get the middle character of a string withouth knowing the string's value. Assume length is odd and >= 1.

```
string = "example"
middle_char = "
```

Hint: `//`

Get the age from this string an int. Do not just say `age = 26`, actually extract it from the string.

```
string = "Age: 26"
age = ____________________
```

Hint: assume age is two characters. Think about what operators are useful here.

Other String Sequence Function

- `string.find(target)`: finds the first index that has the target string, or -1 if not found.

example:

```
"Hello".find("l")   # 2
"Phl".find("U")     # -1
"Attack".find("ta") # 2
```

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

M1:

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 `0.5`, `3.5`, or between those two values. Which is a boolean expression that is `True` only when conditions are safe?

- (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`

The `in` Keyword

In its simplest usage, it checks to see if `something` is within `something else`.

It evaluates to either `True` or `False`. As an example:

```
result = "pressure" in "under_pressure"
print(result) # prints True!!

result = "Maracuyá" in "Mar"
print(result) # prints False!
```

We will eventually see more complex usage; but this is ok for now. :)

Activity: Satisfaction

(S7)

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

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

(S8)

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

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

(S9)

Find a value for `x` such that this expression is `True`, or write "None" if there are none:
Hint: There might be multiple solutions.

```
not ( (3 < x < 8 and x % 2 == 0) and (x // 10 == 0 and x % 2 != 0) )
```

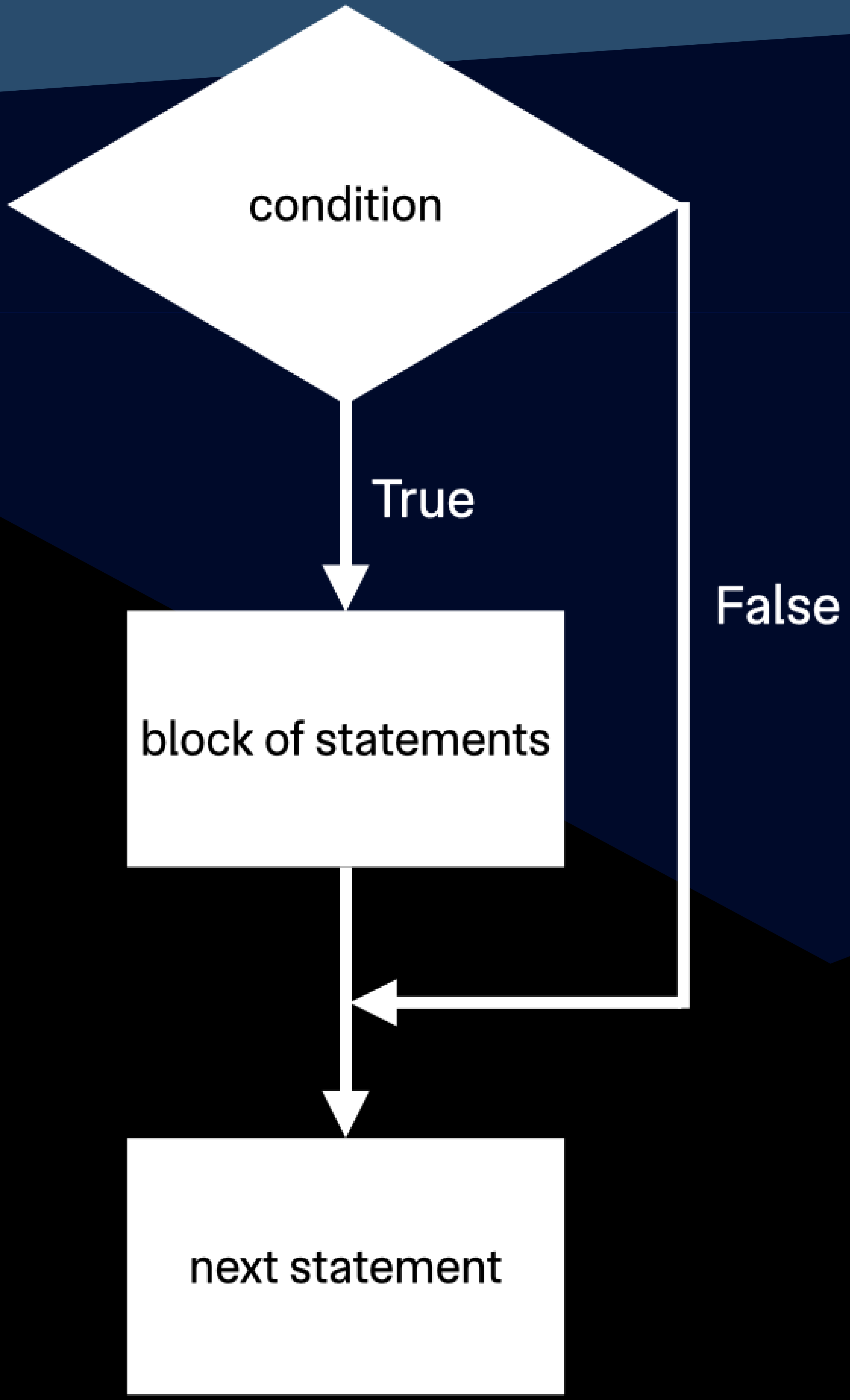
Recap: `if`

"`if` he wanted to, he would." — 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)
```

M2: What color is the circle that gets drawn?

(A) Red

(B) Green

(C) Black

(D) There is no circle drawn

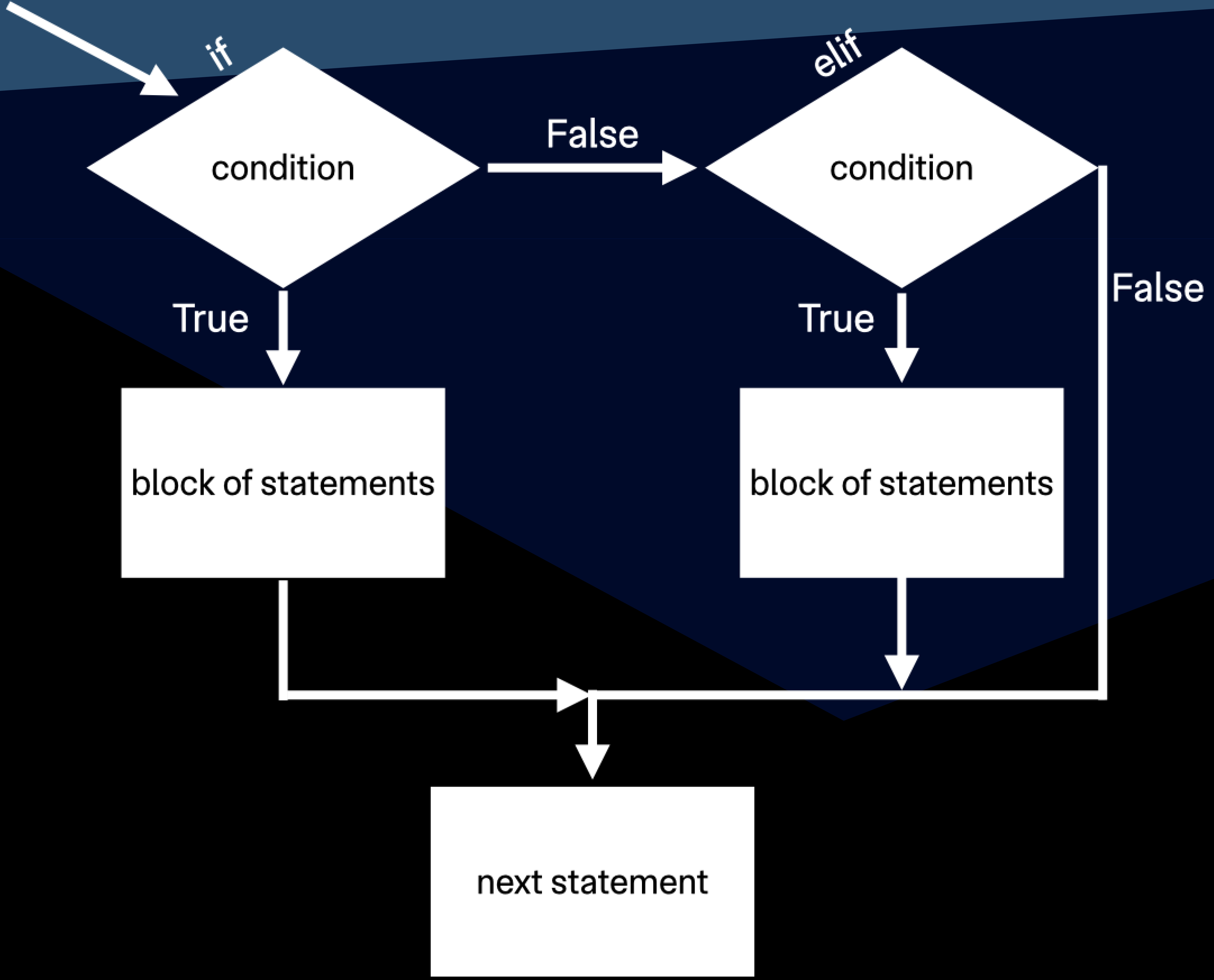
Recap: `elif`

`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*.

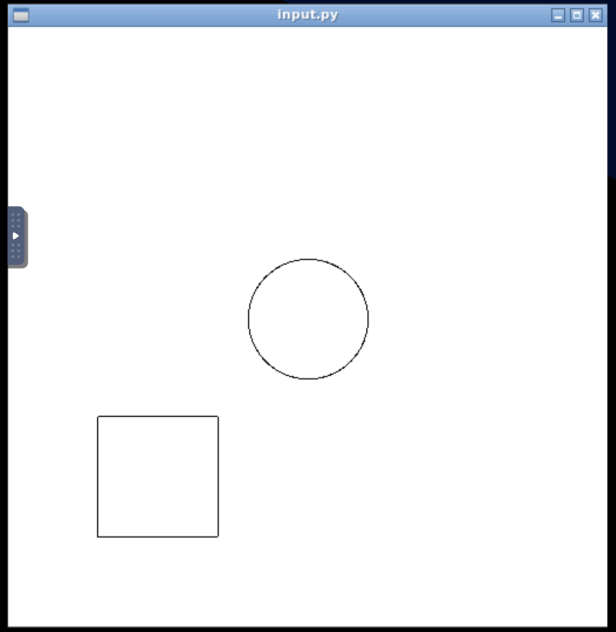
Activity: Picking Paths

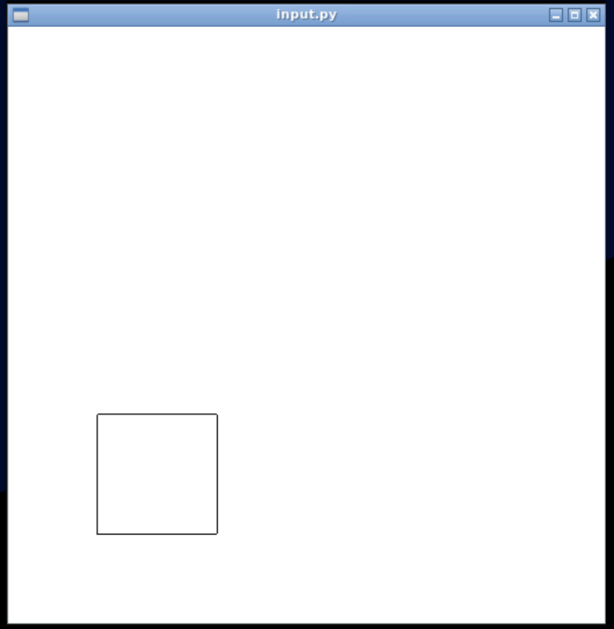
C12: Draw a flow-chart representing the control flow of this program. Use diamonds for conditionals (`if`/`elif`) and put the boolean expressions inside of the diamonds. Represent blocks of code as rectangles. Write all of the lines of code that belong to a block in the rectangle.

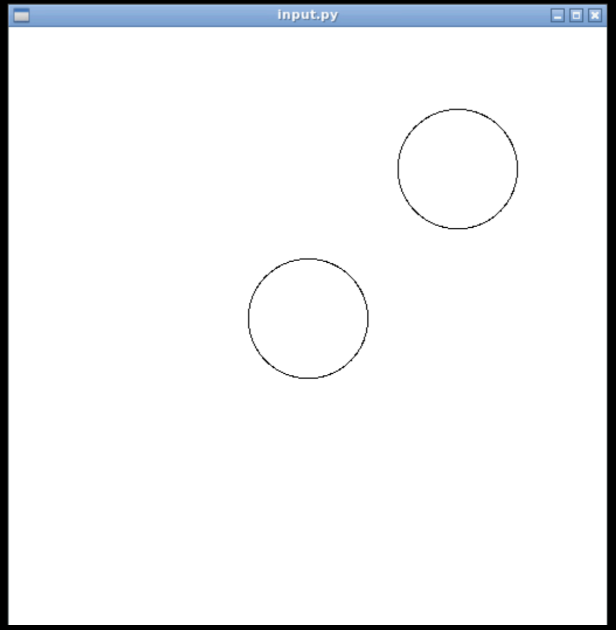
```python
import penndraw as pd
code = input("TYPE THE CODE")

if len(code) == 4:
    pd.square(0.25, 0.25, 0.1)
elif not code.isalpha(): # `isalpha` evaluates to True if all characters are alphabetical...
    pd.circle(0.75, 0.75, 0.1)

if code.endswith("!") and code[0] == "0": # `endswith` evaluates to True if it ends with the str inside ( ).
    pd.circle(0.5, 0.5, 0.1)
pd.run()
```

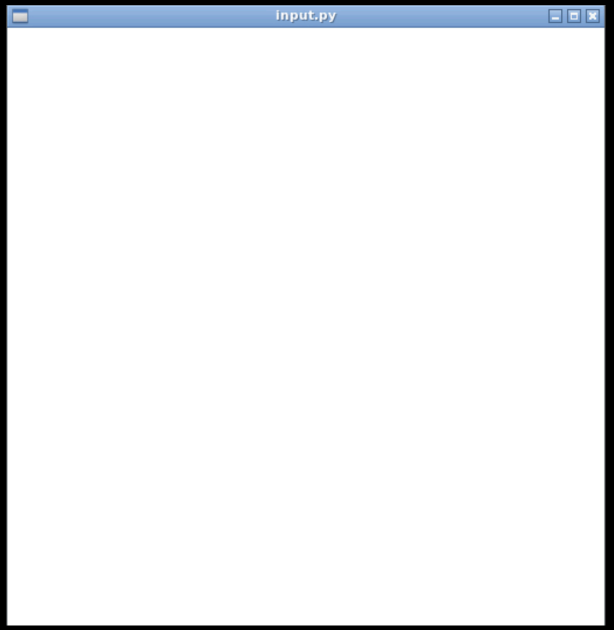
Activity: Picking Paths

Clockwise from the top-left, write the input <"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 if all previous conditions (`if` and `elif`) 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)
```

Extra Practive 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. Here are my personal rankings...:

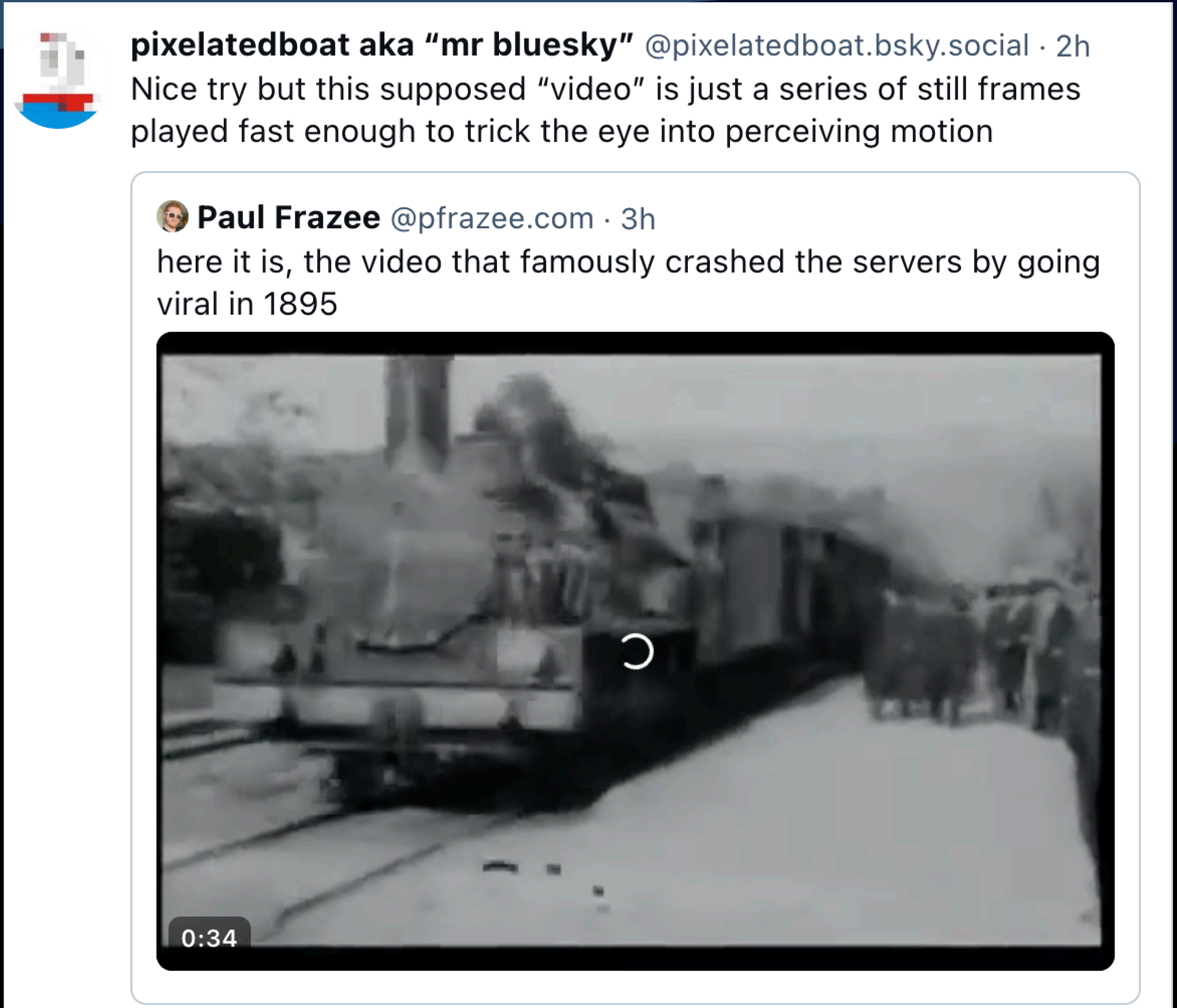
Milks	Tier
Oat, Whole Cow	S (highest)
2% Cow, Soy	A
Coconut, Condensed	B
All other milks	C
Almond, Goat	F (lowest; a disgrace)

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

```python
milk = input()
match milk:
    case "Oat" | "Whole Cow":
        print("S")
    case "2% Cow" | "Soy":
        print("A")
    case "Coconut" | "Condensed":
        print("B")
    case "Almond" | "Goat":
        print("F")
    case _:
        print("C")
```

Reminders

- HW00 is due tonight at 11:59pm
  - Use one late day --> submit by 1/30 @ 11:59pm
  - Use two late days --> submit by 1/31 @ 11:59pm
- Earn late tokens by handing in effortfully completed worksheets
- Check-in due before 1/31
- HW01 Released Tomorrow Afternoon, Due 2/5
  - START EARLY!
- Recitation continues next week



pixelatedboat aka "mr bluesky" @pixelatedboat.bsky.social · 2h

Nice try but this supposed "video" is just a series of still frames played fast enough to trick the eye into perceiving motion

Paul Frazee @pfrazee.com · 3h

here it is, the video that famously crashed the servers by going viral in 1895