Talk on Match Statements!

Intro (~5 mins)

- Hey I'm sam
- school at uts (nearly done)
- coding for most of my life
- pycon a few times

What's wrong with if statements?? (5 minutes)

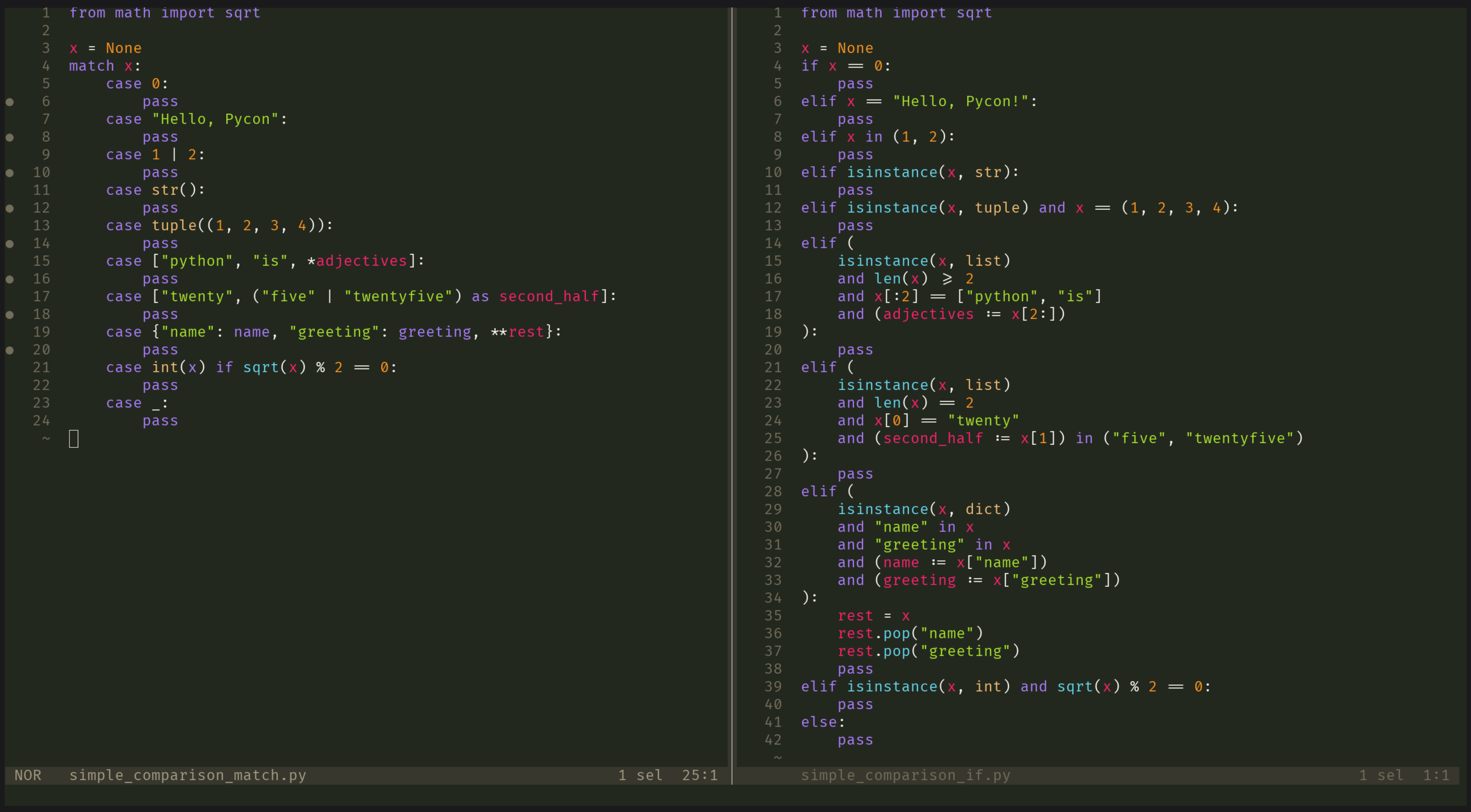
- if statements are pretty good
- sometimes they get pretty bulky
    - here's a quick example we'll go into later, of how *specific* if statements can get pretty long

```
if isinstance(command, list) and len(command)==2 and command[0] == "move": ...
```

- now, you generally wouldn't use one if statement to do all of these things, but then that ends up with greater nesting and, i'd argue, equally hard to traverse code.
- the main takeaway here is, for if statements:
    1. checking types is bulky
    2. checking multiple properties is bulky
    3. they can be hard to read.

intro to match statements

- here's where match statements come in!
- go through the following one by one, match statement first, then it's equivalent if statement
    - note that when pattern matching isn't available, you can still use post-fix if statements to cover missing functionality.



```
from math import sqrt

x = None
match x:
    case 0:
        pass
    case "Hello, Pycon":
        pass
    case 1 | 2:
        pass
    case str():
        pass
    case tuple((1, 2, 3, 4)):
        pass
    case ["python", "is", *adjectives]:
        pass
    case ["twenty", ("five" | "twentyfive") as second_half]:
        pass
    case {"name": name, "greeting": greeting, **rest}:
        pass
    case int(x) if sqrt(x) % 2 == 0:
        pass
    case _:
        pass
```

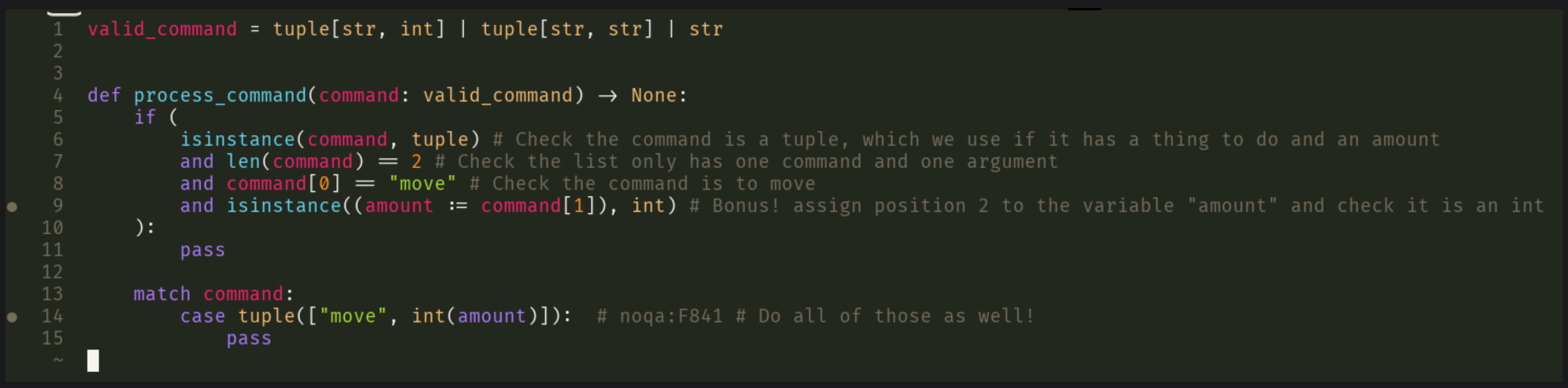
```
from math import sqrt

x = None
if x == 0:
    pass
elif x == "Hello, Pycon!":
    pass
elif x in (1, 2):
    pass
elif isinstance(x, str):
    pass
elif isinstance(x, tuple) and x == (1, 2, 3, 4):
    pass
elif (
    isinstance(x, list)
    and len(x) >= 2
    and x[:2] == ["python", "is"]
    and (adjectives := x[2:])
):
    pass
elif (
    isinstance(x, list)
    and len(x) == 2
    and x[0] == "twenty"
    and (second_half := x[1]) in ("five", "twentyfive")
):
    pass
elif (
    isinstance(x, dict)
    and "name" in x
    and "greeting" in x
    and (name := x["name"])
    and (greeting := x["greeting"])
):
    rest = x
    rest.pop("name")
    rest.pop("greeting")
    pass
elif isinstance(x, int) and sqrt(x) % 2 == 0:
    pass
else:
    pass
```

complex match statements

- so, say it with me now *"how does this play into your talk's premise?"*
- great question everyone, it's because match statements can do more than you think!
- so, example:
    - say you're writing a game in which the player moves around a little dungeon.

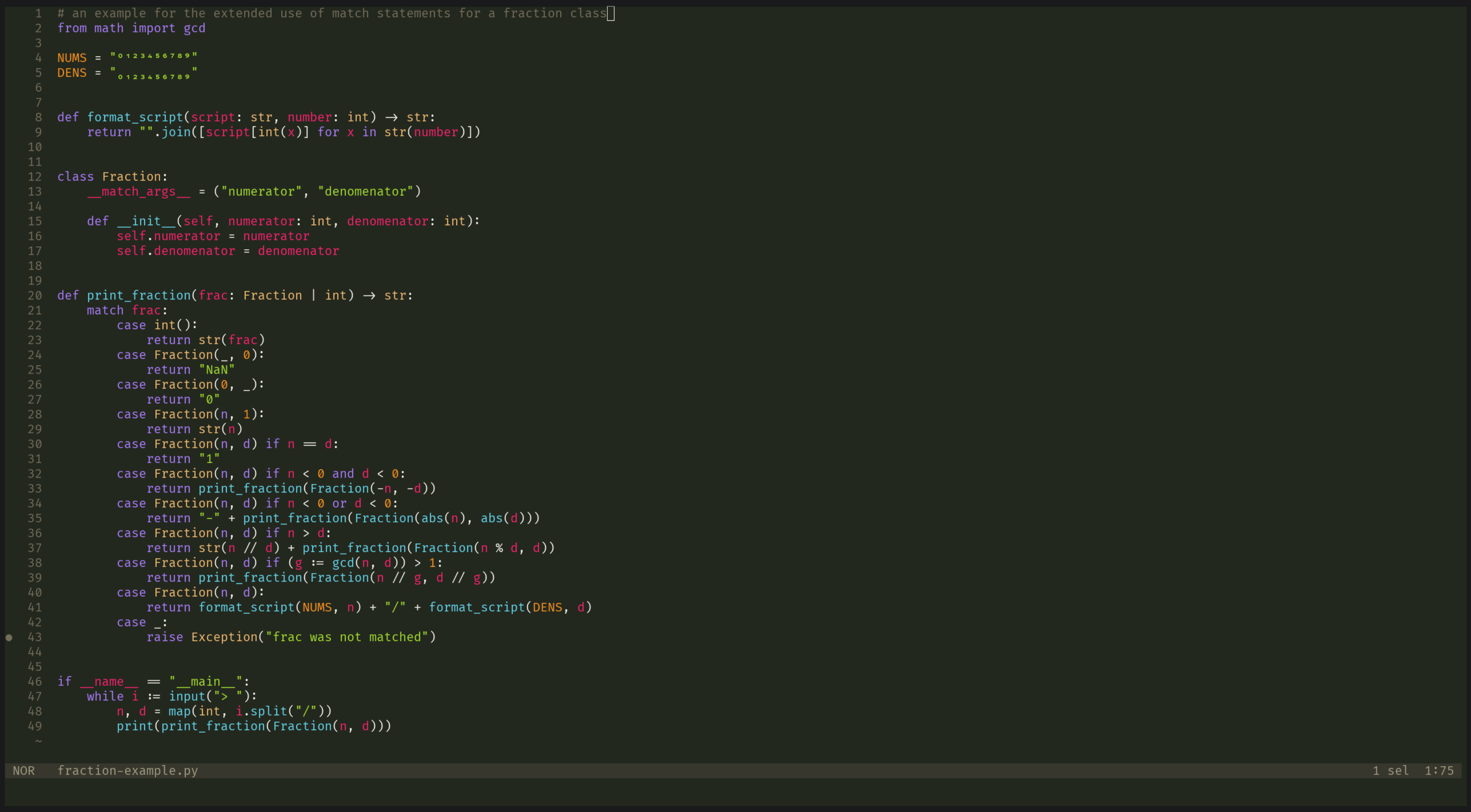
- the player does actions by typing in key words, and amounts, for example: `"turn left"` or `"move once"`
- you already wrote the bit that takes these commands and parses them, but now you need to perform the actions
- here, we get back to our example.

- as I mentioned, this is a bit clunky but in my perfectly normal and non-contrived example, we need a few checks to process the command. This is where Pattern Matching comes in!
- these two blocks of code do exactly the same thing.



```python
valid_command = tuple[str, int] | tuple[str, str] | str


def process_command(command: valid_command) -> None:
    if (
        isinstance(command, tuple) # Check the command is a tuple, which we use if it has a thing to do and an amount
        and len(command) == 2 # Check the list only has one command and one argument
        and command[0] == "move" # Check the command is to move
        and isinstance((amount := command[1]), int) # Bonus! assign position 2 to the variable "amount" and check it is an int
    ):
        pass

    match command:
        case tuple(["move", int(amount)]):  # noqa:F841 # Do all of those as well!
            pass
```

- bonus if i get to it: talk about custom classes and pattern matching there.



```python
# an example for the extended use of match statements for a fraction class
from math import gcd

NUMS = "⁰¹²³⁴⁵⁶⁷⁸⁹"
DENS = "₀₁₂₃₄₅₆₇₈₉"


def format_script(script: str, number: int) -> str:
    return "".join([script[int(x)] for x in str(number)])


class Fraction:
    __match_args__ = ("numerator", "denomenator")

    def __init__(self, numerator: int, denomenator: int):
        self.numerator = numerator
        self.denomenator = denomenator


def print_fraction(frac: Fraction | int) -> str:
    match frac:
        case int():
            return str(frac)
        case Fraction(_, 0):
            return "NaN"
        case Fraction(0, _):
            return "0"
        case Fraction(n, 1):
            return str(n)
        case Fraction(n, d) if n == d:
            return "1"
        case Fraction(n, d) if n < 0 and d < 0:
            return print_fraction(Fraction(-n, -d))
        case Fraction(n, d) if n < 0 or d < 0:
            return "-" + print_fraction(Fraction(abs(n), abs(d)))
        case Fraction(n, d) if n > d:
            return str(n // d) + print_fraction(Fraction(n % d, d))
        case Fraction(n, d) if (g := gcd(n, d)) > 1:
            return print_fraction(Fraction(n // g, d // g))
        case Fraction(n, d):
            return format_script(NUMS, n) + "/" + format_script(DENS, d)
        case _:
            raise Exception("frac was not matched")


if __name__ == "__main__":
    while i := input("> "):
        n, d = map(int, i.split("/"))
        print(print_fraction(Fraction(n, d)))
```