CSC110 Lecture 7: If Statements

Print this handout

Exercise 1: Practice with if statements

1. Consider the code below. Note that the lines are numbered on the left margin.

```
1 def can_vote(age: int) -> str:
2     """Return a string indicating whether age is a legal voting age in Canada.
3
4     In Canada, you must be at least 18 years old to vote.
5     """
6     if age < 18:
7         return 'Too young to vote'
8     else:
9         return 'Allowed to vote'
```

a. In the table below, write down the sequence of line numbers in the function body that will be executed when `can_vote` is called on the given input `age` value.

age	Lines executed
17	1, 6, 7
18	1, 6, 8, 9
19	1, 6, 8, 9

b. Suppose we have the following doctest examples for this function.

```
>>> can_vote(1)
'Too young to vote'
>>> can_vote(1)
'Too young to vote'
```

Why is this not a good choice of doctest examples?

Does not test else part

2. Implement each of the following functions using an if statement.

```
def format_name(given_name: str, family_name: str) -> str:
    """Return the two names as a string of the form '<family_name>, <given_name>'.

    If the family_name is an empty string, return just the given name (without a com

    >>> format_name('Cherilyn', 'Sarkisian')
    'Sarkisian, Cherilyn'
    >>> format_name('Cher', '')
    'Cher'
    """
    ...

def larger_sum(num1: int, num2: int) -> int:
    """Return the list with the larger sum.

    If there is a tie, return num1.

    You may ASSUME that:
    - num1 and num2 are lists of floats.

    >>> larger_sum([1.24, 2.01, 3.3], [3.0, 3.0, 3.0])
    [3.0, 3.0, 3.3]
    >>> larger_sum([2.0, 1.0], [1.0, 2.0])
    [2.0, 1.0]
    """
    ...
```

Exercise 2: If statements with multiple branches

1. Implement each of the following functions, using `elif`s to create if statements with more than two branches.

```
def porridge_satisfaction(temperature: float) -> str:
    """Return what a picky eater says when tasting porridge with the given temperat

    Temperatures greater than 50.0 are too hot, temperatures less than 49.0 are too
    and the temperatures in between are just right.

    >>> porridge_satisfaction(65.5)
    'This porridge is too hot! Ack!'
    >>> porridge_satisfaction(30.3)
    'This porridge is too cold! Brrr...'
    >>> porridge_satisfaction(49.5)
    'This porridge is just right! Yum!'
    """
    ...

def rock_paper_scissors(player1: str, player2: str) -> str:
    """Return the winner of a game of rock, paper, scissors.

    The game is played with the following rules:
        1) 'rock' wins against 'scissors'
        2) 'scissors' wins against 'paper'
        3) 'paper' wins against 'rock'

    Ties are allowed.

    You may ASSUME that the input strings are in {'rock', 'paper', 'scissors'}.

    >>> rock_paper_scissors('rock', 'scissors')
    'Player1 wins'
    >>> rock_paper_scissors('rock', 'paper')
    'Player2 wins'
    >>> rock_paper_scissors('rock', 'rock')
    'Tie!'
    """
    ...
```

Exercise 3: Simplifying if statements

1. Even though this lecture is all about if statements, often we can implement predicates (functions that return booleans) without using if statements at all! For each of the following functions, rewrite the function body so that it does not use an if statement.

```
def is_odd(n: int) -> bool:
    """Return whether n is odd (not divisible by 2).
    """
    if n % 2 == 0:
        return False
    else:
        return True


def is_teenager(age: int) -> bool:
    """Return whether age is between 13 and 18 inclusive.

    HINT: identify the range of integers that make this function return True.
    Your simplified function body can look like:

        return (age >= ...) and (age <= ...)

    or, using a cool Python short-hand:

        return ... <= age <= ...
    """
    if age < 13:
        return False
    else:
        if age > 18:
            return False
        else:
            return True


def larger_first_value(numbers1: list, numbers2: list) -> int:
    """Return the larger of numbers1[0] and numbers2[0].

    You may ASSUME that numbers1 and numbers2 are non-empty lists
    of integers.
    """
    if numbers1[0] >= numbers2[0]:
        return numbers1[0]
    else:
        return numbers2[0]


def is_common_prefix(prefix: str, s1: str, s2: str) -> bool:
    """Return whether prefix is a common prefix of both s1 and s2.

    Hint: identify exactly what boolean expression needs to be True
    in order for this function to return True.
    """
    if str.startswith(s1, prefix):
        if str.startswith(s2, prefix):
            return True
        else:
            return False
    else:
        return False


def same_corresponding_values(mapping: dict, key1: str, key2: str) -> bool:
    """Return whether the two given keys have the same corresponding value in mappi

    Return False if at least one of the keys is not in the mapping.
    """
    if key1 not in mapping:
        return False
    elif key2 not in mapping:
        return False
    elif mapping[key1] == mapping[key2]:
        return True
    else:
        return False
```

Additional exercises

1. *The order of if/elif conditions.*

Consider the following two functions:

```
def pick_animal1(number: int) -> str:
    """Return an animal (based on a number range).
    if number > 1:
        return 'Cat'
    elif number > 10:
        return 'Dog'
    else:
        return 'Duck'


def pick_animal2(number: int) -> str:
    """Return an animal (based on a number range).
    if number > 10:
        return 'Dog'
    elif number > 1:
        return 'Cat'
    else:
        return 'Duck'
```

We will now use our knowledge of comprehensions, `range`, and `all` to write expressions that describe the return values these two functions.

i. Fill in the blanks in each of the following code snippets to satisfy the given description.

a. A set of 5 integers where `pick_animal1` only returns `'Duck'`:

```
>>> all({pick_animal1(x) == 'Duck' for x in _________})
True
```

b. A range of at least 6 integers where `pick_animal1` only returns `'Cat'`:

```
>>> all({pick_animal1(x) == 'Cat' for x in range(___, ___)})
True
```

c. A set of 5 integers where `pick_animal2` only returns `'Duck'`:

```
>>> all({pick_animal2(x) == 'Duck' for x in _________})
True
```

d. The *largest* range of integers where `pick_animal2` only returns `'Cat'`:

```
>>> all({pick_animal2(x) == 'Cat' for x in range(___, ___)})
True
```

e. A range of at least 6 integers where `pick_animal2` only returns `'Dog'`:

```
>>> all({pick_animal2(x) == 'Dog' for x in range(___, ___)})
True
```

2. Can `pick_animal1` ever return `'Dog'`? Why or why not?