Control structures direct the flow of logic in a program. For example, conditionals (if-elif-else) allow a program to skip sections of code, while iteration (while), allows a program to repeat a section.

1.1 If statements

Conditional statements let programs execute different lines of code depending on certain conditions. Let’s review the if-elif-else syntax:

```python
if <conditional expression>:
    <suite of statements>
elif <conditional expression>:
    <suite of statements>
else:
    <suite of statements>
```

Recall the following points:

- The else and elif clauses are optional, and you can have any number of elif clause.
- A conditional expression is a expression that evaluates to either a true value (True, a non-zero integer, etc.) or a false value (False, 0, None, etc.).
- Only the suite that is indented under the first if/elif that has a conditional expression that evaluates to True will be executed.
- If none of the conditional expressions are True, then the else suite is executed. There can only be one else clause in a conditional statement!
1.2 Boolean Operators

Python also includes the boolean operators and, or, and not. These operators are used to combine and manipulate boolean values.

- not returns the opposite truth value of the following expression.
- and short-circuits at the first False value and returns it. If all values evaluate to True, the last value is returned.
- or short-circuits at the first True value and returns it. If all values evaluate to False, the last value is returned.

```python
>>> not None
True
>>> not True
False
>>> -1 and 0 and 1
0
>>> False or 9999 or 1/0
9999
```

1.3 Questions

1. Determine what the Python interpreter will output given the following lines of code.

```python
>>> from operator import add, mul
>>> mul(add(5, 6), 8)
```

Solution: 88

```python
>>> print('x')
```

Solution: x

```python
>>> y = print('x')
```

Solution: x

```python
>>> print(y)
```

Solution: None

```python
>>> print(add(4, 2), print('a'))
```
Solution: a 6 None

def foo(x):
    print(x)
    return x + 1

def bar(y, x):
    print(x - y)

>>> foo(3)

    Solution:
    3
    4

>>> bar(3)

    Solution: Error

>>> bar(6, 1)

    Solution: -5

>>> bar(foo(10), 11)

    Solution:
    10
    0
2. Tommy will only wear a jacket outside if it is below 60 degrees or it is raining.

Write a function that takes in the current temperature and a boolean value telling if it is raining and returns True if Tommy will wear a jacket and False otherwise.

First, try solving this problem using an if statement.
```python
def wears_jacket_with_if(temp, raining):
    """
    >>> wears_jacket_with_if(90, False)
    False
    >>> wears_jacket_with_if(40, False)
    True
    >>> wears_jacket_with_if(100, True)
    True
    """
    Solution:
    if temp < 60 or raining:
        return True
    else:
        return False

Note that we’ll either return True or False based on a single condition, whose truthiness value will also be either True or False. Knowing this, try to write this function using a single line.
```
```python
def wears_jacket(temp, raining):
    Solution:
    return temp < 60 or raining
```

Video walkthrough
3. To handle discussion section overflow, Matt and other TAs may direct students to a more empty section that is happening at the same time.

Write a function that takes in the number of students in two sections and prints out what to do if either section exceeds 30 students.

**Hint:** You can do `str(<number>)+<string>` to concatenate a number and a string

```python
def handle_overflow(s1, s2):
    
    >>> handle_overflow(27, 15)
    No overflow
    >>> handle_overflow(35, 29)
    Move to Section 2: 1
    >>> handle_overflow(20, 32)
    Move to Section 1: 10
    >>> handle_overflow(35, 30)
    No space left in either section

    Solution:
    if s1 <= 30 and s2 <= 30:
        print("No overflow")
    elif s2 > 30 and s1 < 30:
        print("Move to Section 1:" + str(30 - s1))
    elif s1 > 30 and s2 < 30:
        print("Move to Section 2:" + str(30 - s2))
    else:
        print("No space left in either section")
```

**Video walkthrough**
4. Write a function that returns `True` if a positive integer `n` is a prime number and `False` otherwise.

A prime number `n` is a number that is not divisible by any numbers other than 1 and `n` itself. For example, 13 is prime, since it is only divisible by 1 and 13, but 14 is not, since it is divisible by 1, 2, 7, and 14.

**Hint:** use the `%` operator: `x % y` returns the remainder of `x` when divided by `y`.

```python
def is_prime(n):
    ""
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
    ""
```

Solution:

```python
if n == 1:
    return False
k = 2
while k < n:
    if n % k == 0:
        return False
    k += 1
return True
```

Alternatively, the while loop’s conditional expression could ensure that `k` is less than or equal to the square root of `n`.

**Video walkthrough**