For Loops

Learning Goals

- Use for loops when reading and writing algorithms to repeat actions a specified number of times

- Recognize which numbers will be produced by a range expression

- Translate algorithms from control flow charts to Python code

- Use nesting of statements to create complex control flow

For Loops Implement Repeated Actions

We've learned how to use while loops and loop control variables to iterate until a certain condition is met. When that loop control is straightforward (increase/decrease a number until it reaches a certain limit), we can use a more standardized structure instead.

A for loop over a range tells the program exactly how many times to repeat an action. The loop control variable is updated by the loop itself!

```
for <loopVariable> in range(<maxNumPlusOne>):
        <loopBody>
```

While Loops vs. For Loops

To sum the numbers from 0 to n in a while loop, we'd write the following:

```
i = 0
result = 0
while i <= n:
        result = result + i
        i = i + 1
print(result)
```

In a for loop using a range expression, the loop control variable starts at 0 and automatically increases by 1 each loop iteration.

```
result =0
for i in range(n  + 1):
      result = result + i
print(result)
```

We have to use `n + 1` because `range` goes up to but not including the given number. It's like writing

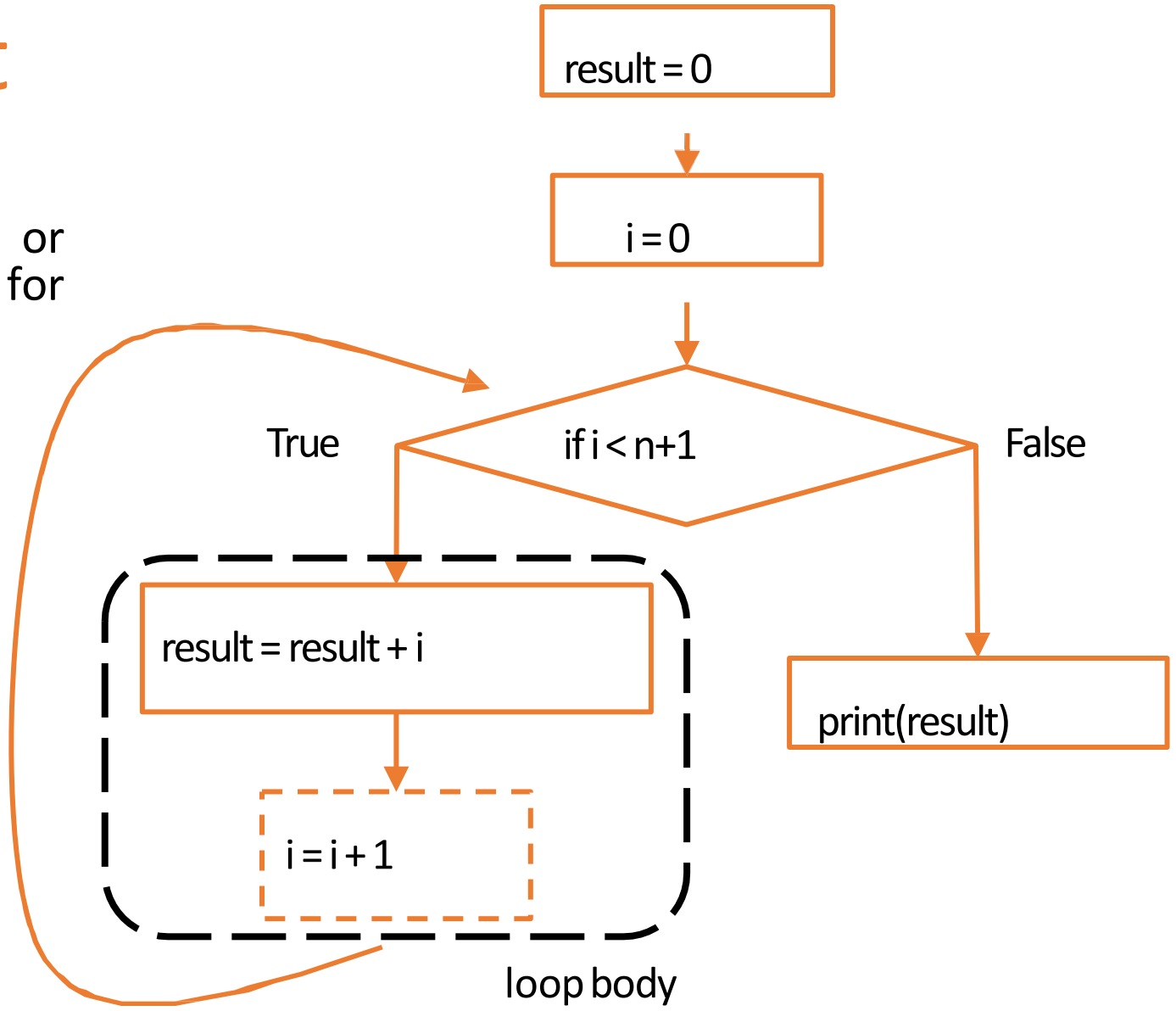
```
while i < n + 1:
```

For Loop Flow Chart

Unlike while loops, we don't initialize or update the loop control variable. The for loop does those actions automatically.

We show actions done by the range function with a dotted outline here, because they're implicit, not written directly.

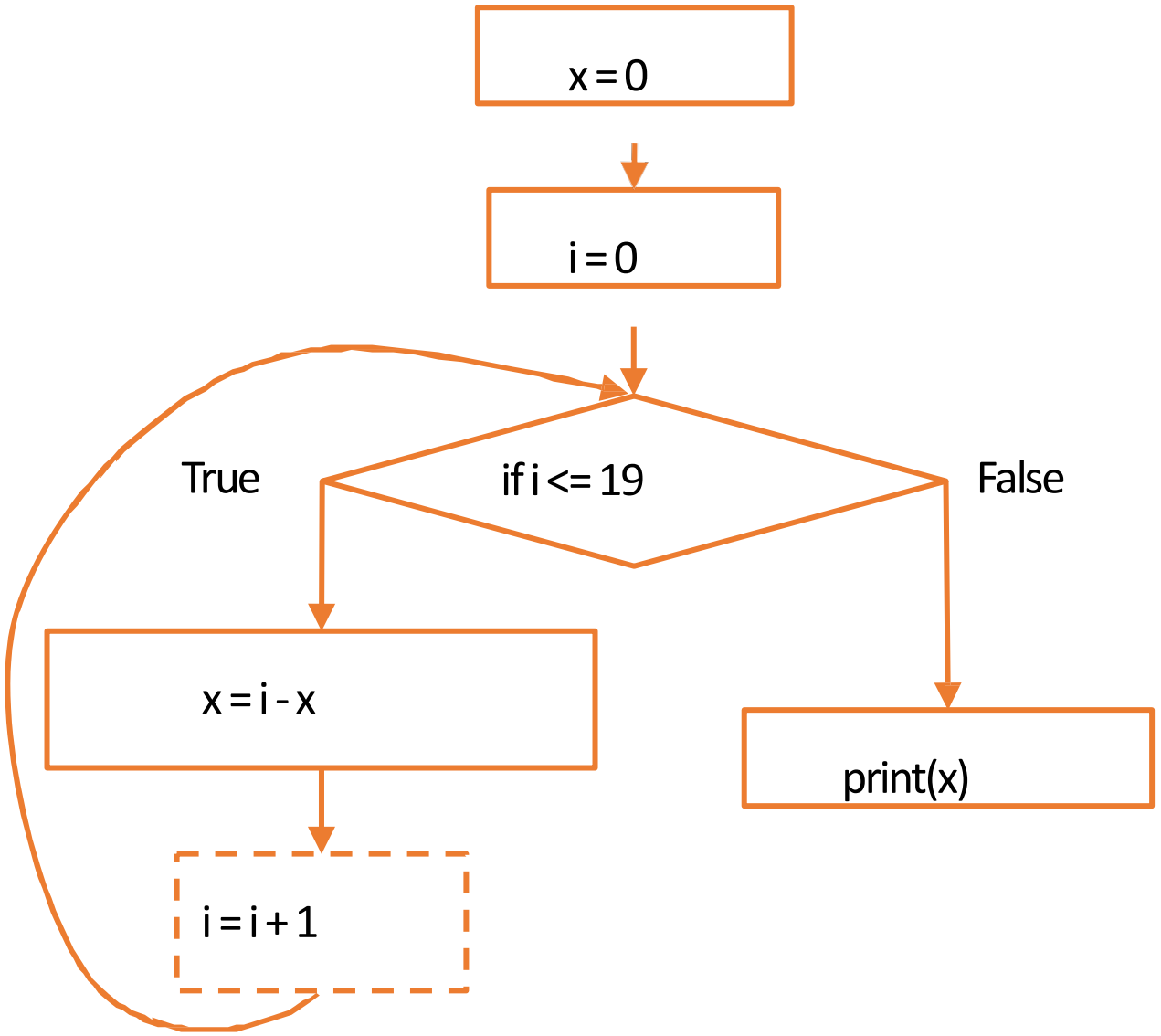
```
result = 0
for i in range(n + 1):
        result = result + i
        print(result)
```



Activity: Translate the Flow Chart

You do: given the flow chart to the right, write a program that matches the flow chart. Use a for loop, not a while loop.

What does the program print?



For Loops Manage the Loop Control Variable

Because the for loop manages the loop control variable, you can't update it in the loop body.

If you try to change the loop control variable, it will revert back to the next expected value on the following iteration. This happens because of the range.

```
for i in range(10):
        print(i)
        i = i + 2 # should skip two ahead, but does not
```

Range

range() Generates Loop Variable Values

When we run for i in range(10), range(10) generates the consecutive values 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 for the loop control variable, one value each iteration.

We can also give range() two arguments, a start and an end value. The loop control variable begins with the start value, is incremented by 1 each iteration, and goes up to but not including the end value.

The following code would generate the numbers 3, 4, 5, 6, and 7.

```
for i in range(3, 8):
        print(i)
```

range() Also Has a Step

If we use three arguments in the range()function, the last argument is the step of the range (how much the loop control variable should change in each iteration). The following example would print the even numbers from 1 to 10, because it updates i by 2each iteration.

```
for i in range(2, 11, 2):

        print(i)
```

Anything we can do in a for loop can also be done in a while loop. In a while loop, the above code could be written as:

```
i = 2
while i < 11:
        print(i) i = i + 2
```

range() Example: Countdown

Let's write a program that counts backwards from 10 to 1, using range().

```
for i in range(10, 0, -1):
        print(i)
```

Note that i has to end at 0 in order to make 1 the last number that is printed.

Problem Solving with For Loops

Problem solving with for loops is similar to problem solving with while loops. You need to identify the loop control variable, then find the correct start, end, and step for it.

Example: how would you create a program that produces the pattern "10-11-12-13-" using a for loop?

```
s = ""
for i in range(10, 14):
        s = s + str(i) + "-"
print(s)
```

Nested Loops

Nesting Loops

We've already used nesting to put conditionals in other conditionals and in loops. Importantly, we can also nest loops inside of loops!

We mostly do this with for loops, and mostly when we want to loop over multiple dimensions.

```
for <loopVar1>      in   range(<endNum1>):
        for <loopVar2>    in range(<endNum2>):
            <bothLoopsBody>
        <justOuterLoopBody>
```

In nested loops, the inner loop is repeated every time the outer loop takes a step.

Example: Coordinate Plane with Nested Loops

Suppose we want to print all the coordinates on a plane from (0,0) to (4,3).

```
for x in range(5):
        for y in range(4):
                print("(", x, ",", y, ")")
```

Note that every iteration of y happens anew in each iteration of x.

Tracing Nested Loops

The following code prints out a 4x3 multiplication table. We can use code tracing to find the values at each iteration of the loops.

```
for x in range(1, 5):
    for y in range(1, 4):
        print(x, "*", y, "=", x * y)
```

Iteration	x	y	x*y
1	1	1	1
2	1	2	2
3	1	3	3
4	2	1	2
5	2	2	4
6	2	3	6
7	3	1	3
8	3	2	6
9	3	3	9
10	4	1	4
11	4	2	8
12	4	3	12