

While Loops

15-110 – Friday 09/13

Announcements

- Check2 due **Monday at noon**
- You're encouraged to attend **small group sessions** to get help with learning the course content. In particular, the TAs will provide **more help than usual** on one of the Hw2 problems in small group sessions next week ([drawIllusion](#)). Contact your TA to learn more!
- Check1/Hw1 revision deadline: **Tuesday 09/17 noon**
 - If you want to update your submission based on feedback, just make the changes to your solution and resubmit (applies to exercises too!)
 - TAs will regrade within a week, usually
 - Note that revision submissions are capped at **90 points** – don't resubmit if you already scored a 90 or above. (Still look at your feedback, though!)

Learning Goals

- Use **while loops** when reading and writing algorithms to repeat actions while a certain condition is met
- Identify **start values, continuing conditions, and update actions** for **loop control variables**
- Translate algorithms from **control flow charts** to Python code
- Use **nesting** of statements to create complex control flow

Repeating actions is messy.

Let's write a program that prints out the numbers from 1 to 10.
Up to now, that would look like:

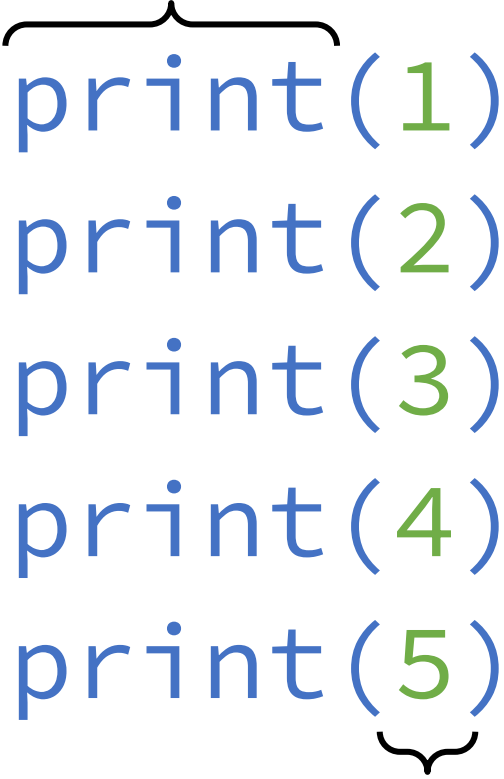
```
print(1)
print(2)
print(3)
print(4)
print(5)
print(6)
print(7)
print(8)
print(9)
print(10)
```

A **loop** is a **control structure** that lets us repeat actions so that we do not need to write out similar code over and over again.

Loops are powerful when we want to **repeat a pattern**.

Identify the parts of the pattern that are the same and the parts that are different.

same



The diagram shows five lines of code: `print(1)`, `print(2)`, `print(3)`, `print(4)`, and `print(5)`. A horizontal curly brace above the first three lines is labeled 'same' in red. A vertical curly brace below the last three lines is labeled 'different' in red. The numbers 1 through 5 are highlighted in green.

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

different

While Loops

A **while loop** is a type of loop that keeps repeating only while a certain condition is **True**.

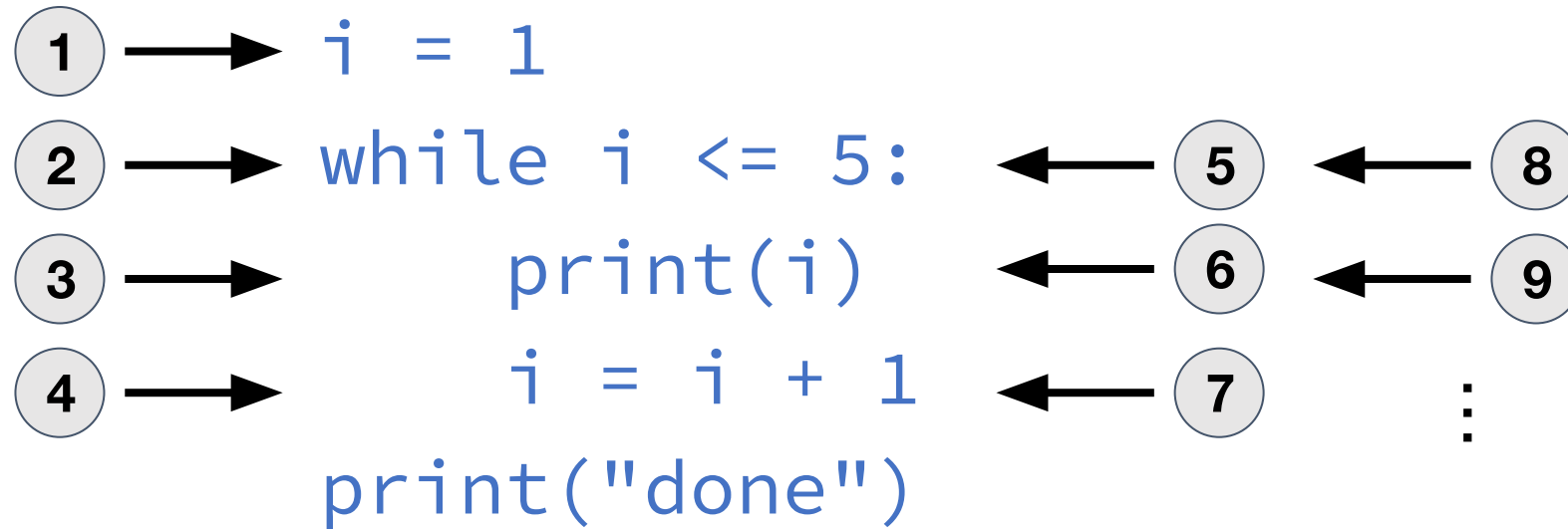
```
while <booleanExpression>:  
    <loopBody>
```

indentation
(tab) →

```
1 i = 1  
2 while i <= 5:  
3     print(i)  
4     i = i + 1
```

while is how Python knows this is a while loop
: and **indentation** is start of **while loop body**

The `while` statement body **must become False** for the loop to end.



Check this out on pythontutor.com.

A `while` loop with a condition that does not become `False` is an **infinite loop**.

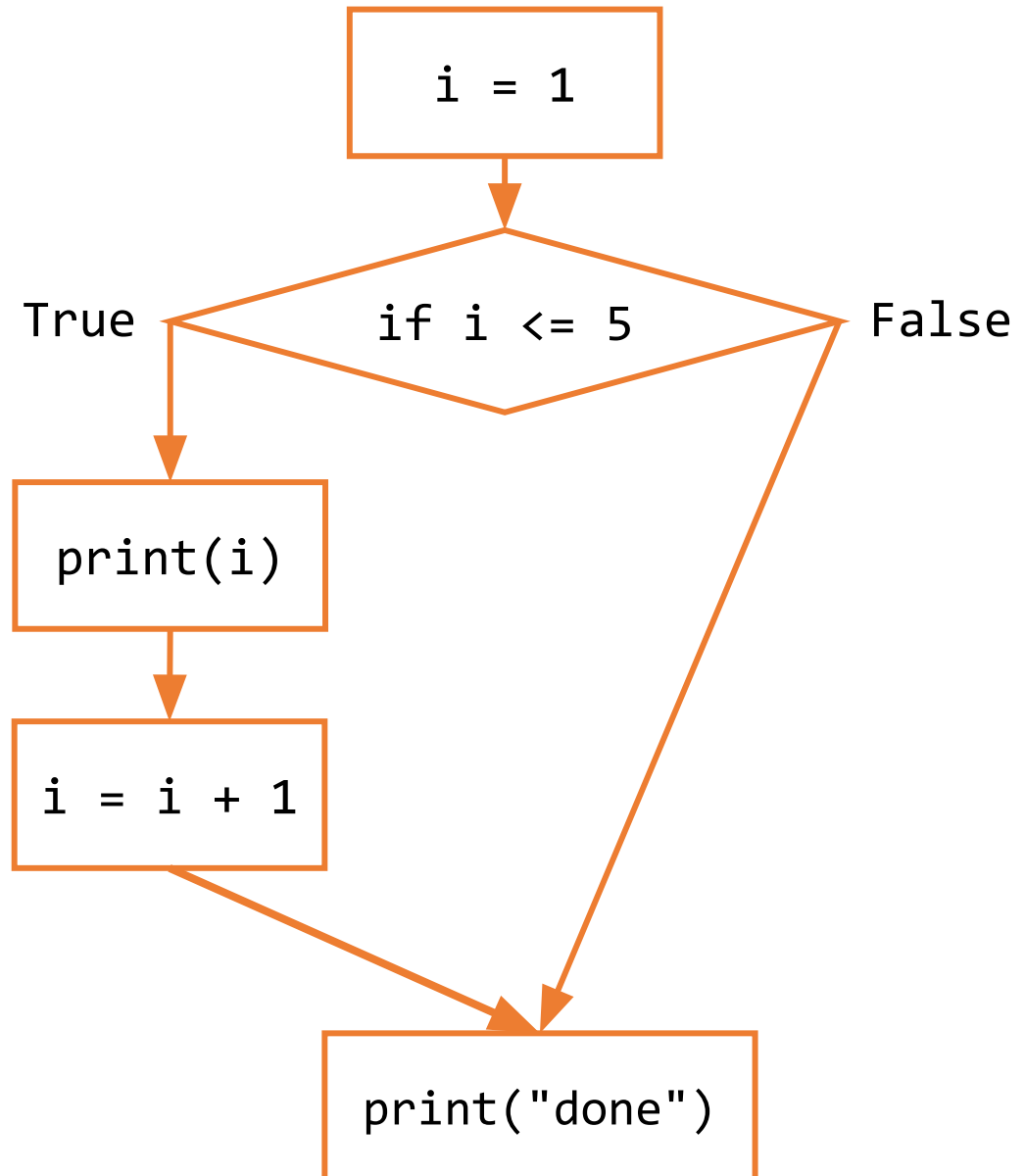
```
i = 1
while i > 0:
    print(i)
    i = i + 1
```



Hit the **STOP button** in Thonny to stop an infinite loop!

if Flow Chart

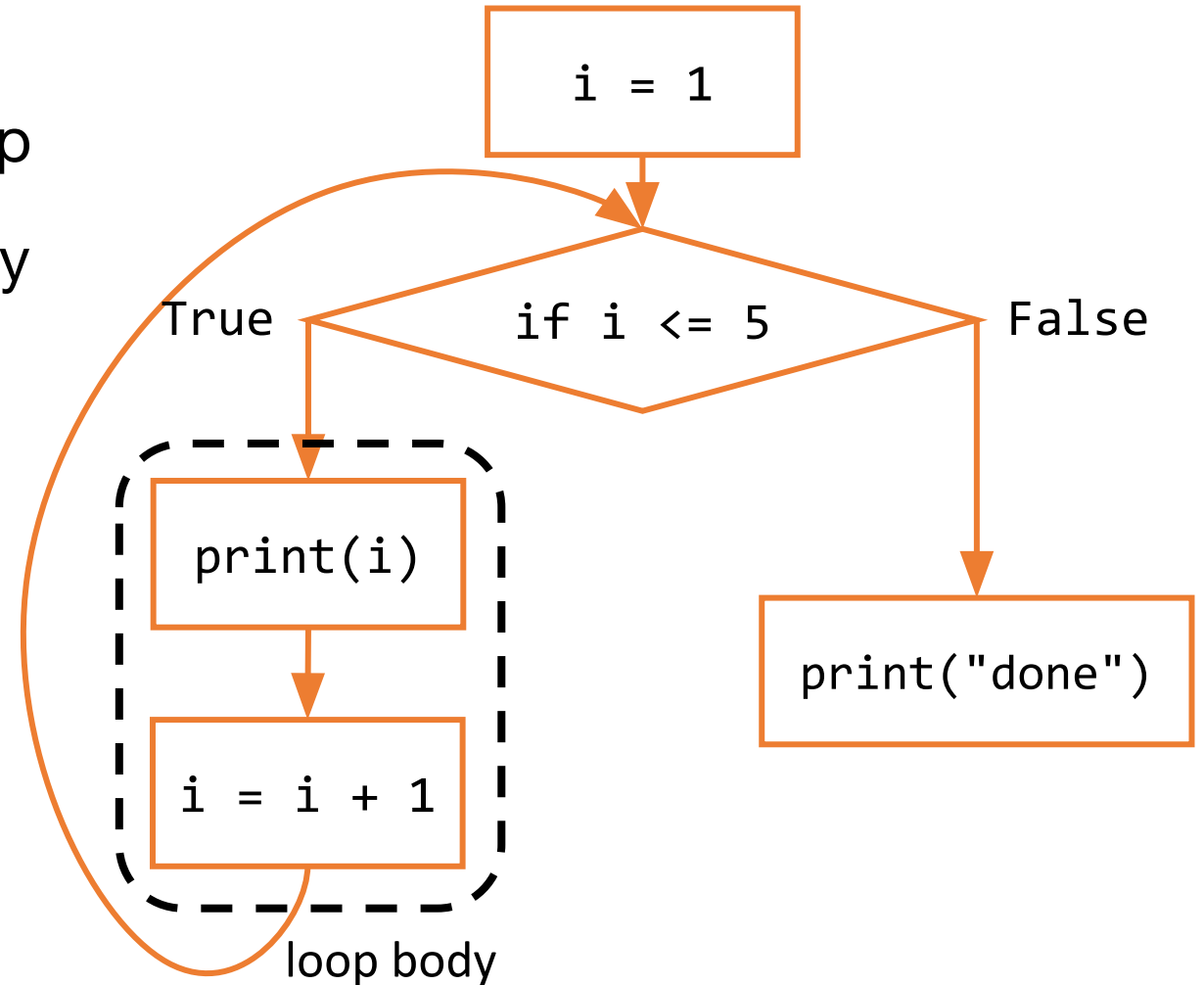
```
i = 1
if i <= 5:
    print(i)
    i = i + 1
print("done")
```



while Loop Flow Chart

Unlike an `if` statement, a `while` loop flow chart needs to include a transition from the `while` loop's body back to itself.

```
i = 1
while i <= 5:
    print(i)
    i = i + 1
print("done")
```



Activity: Trace the Program

You do: if we slightly change the code from the previous program, what happens to the program?

```
i = 1
while i <= 5:
    i = i + 1 # moved up one line
    print(i)
print("done")
```

Loop Control Variables

To design algorithms with loops, we need to identify what needs to change each iteration by creating a **loop control variable**.

A loop control variable **needs three things to work correctly**: start value, continuing condition, update action

Algorithm: Print numbers 1 to 10 (inclusive)

control variable: `num`

start value: `1`

continuing condition: `num <= 10`

update action: `num = num + 1`

```
num = 1
while num <= 10:
    print(num)
    num = num + 1
```

To **count backwards** we change the start value, continuing condition, and update action of the loop control variable.

Algorithm: Print numbers 10 to 1 (inclusive)

control variable: `num`

start value: `10`

continuing condition: `num >= 1`

update action: `num = num - 1`

```
num = 10
while num >= 1:
    print(num)
    num = num - 1
```

Activity: Print Even Numbers

You do: your task is to print the even numbers from 2 to 100 (inclusive).

What is your loop control variable? What is its start value, continuing condition, and update action?

Once you've determined what these values are, use them to write a short program that does this task.

Loops in Algorithms

We can update additional variables inside a loop body to implement **more complex algorithms**.

Algorithm: Add the numbers from 1 to 10

Need to keep track of two numbers:

- the current number we are adding
- the current sum

```
result = 0
num = 1
while num <= 10:
    result = result + num
    num = num + 1
print(result)
```

Which is the loop control variable?

It is important to be able to **trace loops** to understand (and debug!) what a program is doing at the **end of each iteration**.

```
result = 0
num = 1
while num <= 7:
    result = result + num
    num = num + 1
print(result)
```

	result	num
before loop	0	1
iteration 1	1	2
iteration 2	3	3
iteration 3	6	4
iteration 4	10	5
iteration 5	15	6
iteration 6	21	7
iteration 7	28	8
after loop	28	8

Order matters when updating variables and Python only checks the loop condition at the end of each iteration.

```
result = 0
num = 1
while num <= 7:
    num = num + 1
    result = result + num
print(result)
```

	result	num
before loop	0	1
iteration 1	2	2
iteration 2	5	3
iteration 3	9	4
iteration 4	14	5
iteration 5	20	6
iteration 6	27	7
iteration 7	35	8
after loop	35	8

We can **nest conditionals** in `while` loops, putting conditions on control variables.

Algorithm: (printing patterns)
Write code to produce the following

x-x-x
-o-o-
x-x-x
-o-o-
x-x-x

If the control variable is even,
print x, if odd print o.

```
row = 0
while row < 5:
    if row % 2 == 0:
        print("x-x-x")
    else:
        print("-o-o-")
    row = row + 1
```

We can nest `while` loops in functions.

If we `return` inside a loop, Python immediately exits the function – no further iterations will run.

Algorithm: Check whether a multiple of num occurs within a certain range [start, end]

```
def multipleInRange(start, end, num):  
    i = start  
    while i <= end:  
        print(i) # shows loop ends early  
        if i % num == 0:  
            return True  
        i = i + 1  
    return False
```

Coding with Multiple Data Points

Now that we have loops, we can start writing algorithms to solve real-world problems. For example, we often want to analyze multiple data points while writing code.

Loops make it possible for us to repeat an action multiple times- that should make it possible for us to get multiple data points. But how can we receive that data?

For now, we'll use the `input` built-in function to repeatedly ask the user for data. Later we'll learn about a new data type that can store multiple values in one place.

Looping with input

If we call `input` inside the loop body, we can get multiple inputs from the user and process them like a data stream.

We'll need to give the user a way to signal that they're done entering numbers. This can be done with a special input, like the string `'q'`.

For example, this code sums the numbers entered by the user until they signal an end to the numbers.

```
result = 0
value = input("Enter a number, or q to quit:")
while value != "q":
    num = int(value)
    result = result + num
    value = input("Enter a number, or q to quit:")
print("Total sum:", result)
```

Note: our loop control variable here is `value`. It starts as a user input, is updated by asking for new input, and continues looping while it is not `"q"`.

Learning Goals

- Use **while loops** when reading and writing algorithms to repeat actions while a certain condition is met
- Identify **start values, continuing conditions, and update actions** for **loop control variables**
- Translate algorithms from **control flow charts** to Python code
- Use **nesting** of statements to create complex control flow

Extra Slides:

Advanced Loops in Algorithms

This content will not be tested, but is interesting to know!

Loop Control Variables – Advanced Example

It isn't always obvious how the start values, continuing conditions, and update actions of a loop control variable should work. Sometimes you need to think through an example to make it clear!

Example: simulate a zombie apocalypse. Every day, each zombie finds and bites a human, turning them into a zombie. If we start with just one zombie, how long does it take for the whole world (7.5 billion people) to turn into zombies?

We'll need to track and update **two** variables- one for the number of zombies, one for the number of days passed.

Loop control variable: # of zombies

Start value: 1 zombie

Continuing condition: while the number of zombies is less than the population

Update action: double the number of zombies every day

```
zombieCount = 1
population = 7.5 * 10**9
daysPassed = 0
while zombieCount < population:
    daysPassed = daysPassed + 1
    zombieCount = zombieCount * 2
print(daysPassed)
```

Loop Control Variables – Another Example

Example: how would you count the number of digits in an integer?

One answer: A number **abc** can be written as:

$$a*100 + b*10 + c*1$$

or

$$a*10^2 + b*10^1 + c*10^0$$

Check each power of 10 until one is bigger than the number. A separate variable can track the actual number of digits counted.

Loop control variable: which power of 10 is being checked

Start value: 1 (10^0)

Continuing condition: while the power of 10 isn't greater than the number

Update action: multiply the power by 10

```
num = 2021
power = 1
digits = 0
while power < num:
    digits = digits + 1
    power = power * 10
print(digits)
```

Loop Control Variables – Another Example

Another answer: instead of comparing a power of 10 to the number, change the number itself.

For example, to count the digits in **abc**, change:

abc ->

ab ->

a

The number of times you can **divide the number by 10** is the number of digits.

Loop control variable: the number itself

Start value: the number's initial value

Continuing condition: while the number is not yet 0 (no digits)

Update action: divide the number by 10

```
num = 2021
```

```
digits = 0
```

```
while num > 0:
```

```
    digits = digits + 1
```

```
    num = num // 10
```

```
print(digits)
```