

# Looping



# Repetition Structure

- Sometimes you need to repetitively execute some code. For example.

```
2  
3 print("Beetlejuice")  
4 print("Beetlejuice")  
5 print("Beetlejuice")  
6
```

- Not a difficult issue, but what if you wanted to print the same code *100 times* or even more
- **Don't Repeat Yourself** – DRY Principle of Coding

# Repetition Structure

- The repetition structure is also called a loop
- A loop is a section of programming instructions that are performed –
  - perhaps repeatedly and as many times as is needed
- Looping has the following advantages:
  - Use of one set of instructions to process multiple sets of instructions
  - Include selection structures to allow treating different data in varying ways, depending on the situation
  - Operate on know or unknown quantities

# Repetition Structure

- To determine how many loops to execute (or even if they need to be executed) is determined by a conditional statement
- The difference between a selection structure and a loop lies in the words **perhaps repeatedly**
- *After* the statement in a loop are performed, the condition is evaluated again and **depending** on the evaluation's outcome the statement *may* be performed again
- Each execution of the statement is called an **iteration**

# Repetition Structure

- A condition is a comparison of a variable with a value – in a loop this variable is called a loop variable
- There are three components in a loop that must be included:
  - **Initialization** – an initial value is assigned to the loop variable
  - **Condition evaluation** – a condition is evaluated that determines whether the loop iterates
  - **Alteration** – the loop variable can be changed so that the condition is eventually evaluated differently and the loop can terminate
  - **NOTE** – without allowing for the changing variable you can get caught in an **infinite loop** – a loop that never stops

# Repetition Structure

- Loops normally perform a specific operation a set number of times. To ensure that the operation is performed the specified number of times, a counter is required. A counter is numeric and is often named index
- **Incrementing** the counter will add value to the number each time the loop completes

# While Loop

- The While loop is a common repetition structure
- It consists of three parts:
- Loop variable (**sentinel value**) is initialized before loop starts
- In loop header, the word while is followed by a condition
  - If condition evaluates **true**, statements in loop body are performed until the end of the loop is reached
  - If condition is **false**, the program skips to statements after loop
- Loop variable is altered somewhere in the loop body
  - Allows the condition to eventually become false
- This type of loop is called a pretest loop since the condition is tested BEFORE the loop is executed

# While Loop

```
3 sentinelValue = 1
4
5 while sentinelValue <= 3:
6     print("Beetlejuice")
7     sentinelValue += 1
8
9 |
```



# For Loop

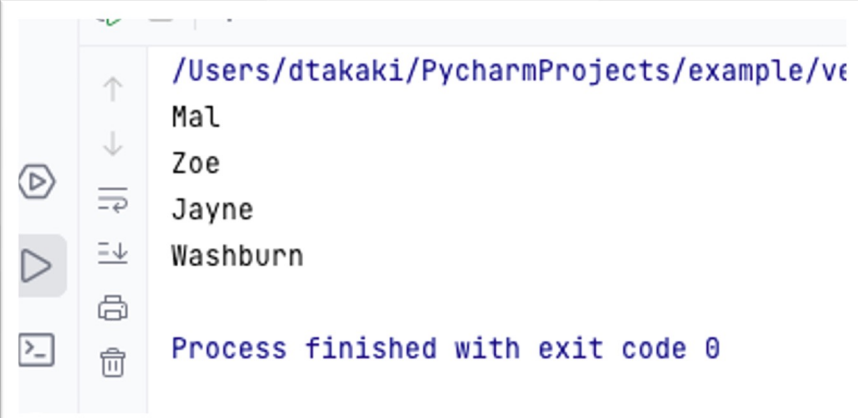
- A common task is to access all elements in a container
- A **for loop** iterates over each element in a container
  - No sentinel value is required - it only executes as many times as there are items in the container
  - Syntax is

```
for variableName in container:  
    code to execute
```

# For Loop

Variable that will be assigned  
each element in the container

```
2  
3 students = ["Mal", "Zoe", "Jayne", "Washburn"]  
4  
5 for student in students:  
6     print(student)  
7
```

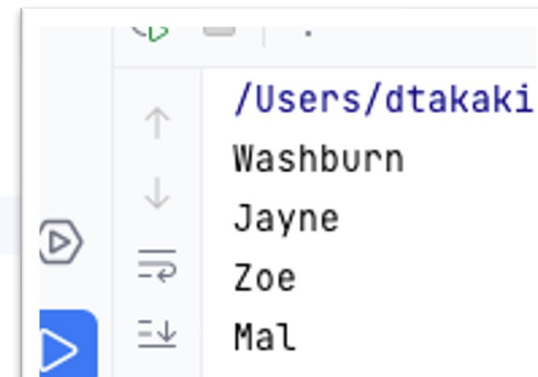


```
/Users/dtakaki/PycharmProjects/example/ve  
Mal  
Zoe  
Jayne  
Washburn  
  
Process finished with exit code 0
```

# For Loop

- By default a loop starts at the beginning of the collection and iterates over each until the end
- The order of elements can be reversed, using the `reversed()` function
  - Items will be iterated over from the end to the beginning.

```
1  
2  
3 students = ["Mal", "Zoe", "Jayne", "Washburn"]  
4  
5 for student in reversed(students):  
6     print(student)  
7
```





# Ranges

- The `range()` function allows counting in for loops
  - It generates a sequence of integers between the starting integer (inclusive) to an ending integer (exclusive) at a specific interval (step value)

# Ranges

Range	Generated sequence	Explanation
<code>range(5)</code>	<code>0 1 2 3 4</code>	Every integer from 0 to 4
<code>range(0, 5)</code>	<code>0 1 2 3 4</code>	Every integer from 0 to 4
<code>range(3, 7)</code>	<code>3 4 5 6</code>	Every integer from 3 to 6
<code>range(10, 13)</code>	<code>10 11 12</code>	Every integer from 10 to 12
<code>range(0, 5, 1)</code>	<code>0 1 2 3 4</code>	Every 1 integer from 0 to 4
<code>range(0, 5, 2)</code>	<code>0 2 4</code>	Every 2nd integer from 0 to 4
<code>range(5, 0, -1)</code>	<code>5 4 3 2 1</code>	Every 1 integer from 5 to 1
<code>range(5, 0, -2)</code>	<code>5 3 1</code>	Every 2nd integer from 5 to 1

# Ranges

```
3
4 initial_savings = 100
5 interest_rate = 0.05
6
7 years = int(input('Enter years: '))
8 print()
9
10 savings = initial_savings
11 for i in range(years):
12     print(f'Savings in year {i}: ${savings:.2f}')
13     savings = savings + (savings*interest_rate)
14
15 print('\n')
16
```

Enter years: 5

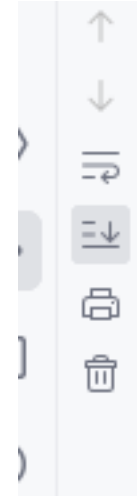
Savings in year 0: \$100.00  
Savings in year 1: \$105.00  
Savings in year 2: \$110.25  
Savings in year 3: \$115.76  
Savings in year 4: \$121.55


# Nested Loops

- A nested loop is a loop that appears in the body of another loop
  - The outer loop is the main loop
  - The inner loop is the nested loop
- Inner loops will complete **all** of their iterations for **every single** iteration of the outer loop

# Nested Loops

```
for hours in range(1):  
    for minutes in range(60):  
        for seconds in range(60):  
            print(f'{hours}:{minutes}:{seconds}')
```



0:0:0  
0:0:1  
0:0:2  
0:0:3  
0:0:4  
0:0:5  
0:0:6  
0:0:7  
  
0:59:52  
0:59:53  
0:59:54  
0:59:55  
0:59:56  
0:59:57  
0:59:58



# Break statement

- The break statement is used to immediately exit a loop
- If I want to display the numbers from 1 - 100 that are **even**, I could use this code.

```
2  
3 for number in range(1,101):  
4     if number % 2 == 1:  
5         break  
6     print("This will never run")  
7
```

Break stops the code from completing its task

# Continue

- The continue statement causes the code to not perform the next line of code, but **continue** to the next iteration loop

```
7
10 for number in range(1,101):
11     if number % 2 == 1:
12         continue
13     print(f'{number} is an even number')
14
```

```
/Users/dtakaki/PycharmProjects/exam
2 is an even number
4 is an even number
6 is an even number
8 is an even number
10 is an even number
12 is an even number
14 is an even number
...
```

# Loop else

- A loop else provides code that can be run once the loop has ended.
  - This is only run if the code does not use a break as this is still part of the overall loop



# Value and Index

- There are times where you want to get the value and its location (index) in the container
- The `enumerate()` function retrieves the values and creates a tuple with the (index, value)

# Enumerate()

```
3  
4 students = ["Mal", "Simon", "Zoe", "River"]  
5  
6 for (index, student) in enumerate(students):  
7     print(f'{student} is located at index {index}')  
8  
9
```

```
/Users/dtakaki/PycharmProjects/exam  
Mal is located at index 0  
Simon is located at index 1  
Zoe is located at index 2  
River is located at index 3  
  
Process finished with exit code 0
```