Guide to For Loops

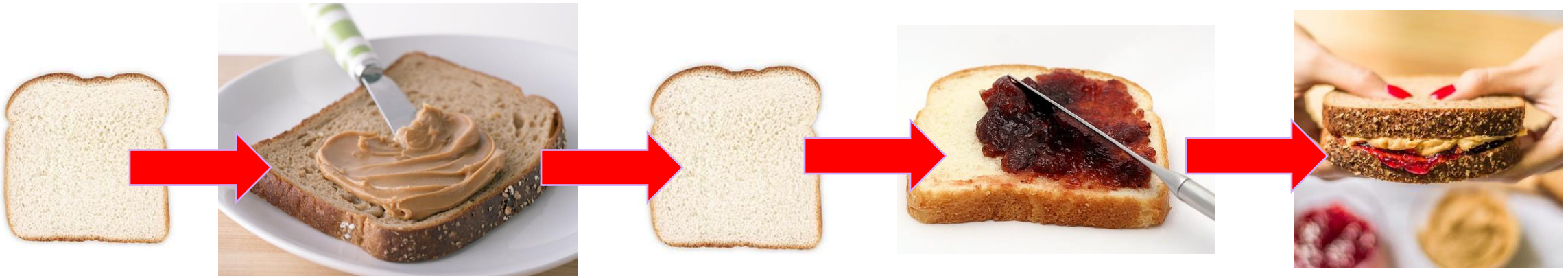
PRESENTED BY: LINDA, JULIE, JASMINE, AND CORNELIUS

Why Use Loops?

- Idea: we (the coders) want to do as little work as possible
- Example: I am hungry, and I love PBJs.
- I want 500 PBJs
- Steps to making PBJ:
  - Get slice of bread
  - Put peanut butter on bread
  - Grab another slice of bread
  - Put jelly on this slice
  - Smash together

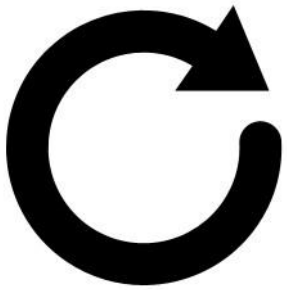
Why Use Loops?

- We could repeat the PBJ steps 500 times
  - Lot of work on our part...
- Or, we could write a program that makes one PBJ sandwich and then tell something to run that program 500 times
  - Less work on our part!



Why Use Loops?

- Same idea in a pure coding setting
- If we want to run the same bit of code on perhaps different inputs (so, make the same type of sandwich but start with different pieces of bread), we can use a loop
  - Write the code to do the task
  - Tell Python to loop and repeat the task many times



What Can We Loop Over?

In short, we can loop over anything that is an "iterable"

Lists are iterable

```
>>> the_list = [10,11,12,13]
>>> for x in the_list:
...     print(x)
...
10
11
12
13
>>>
```

Strings are iterable

```
>>> the_string = "Hello!"
>>> for x in the_string:
...     print(x)
...
H
e
l
l
o
!
>>>
```

Integers are not

```
>>> for x in 5:
...     print(x)
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
>>>
```

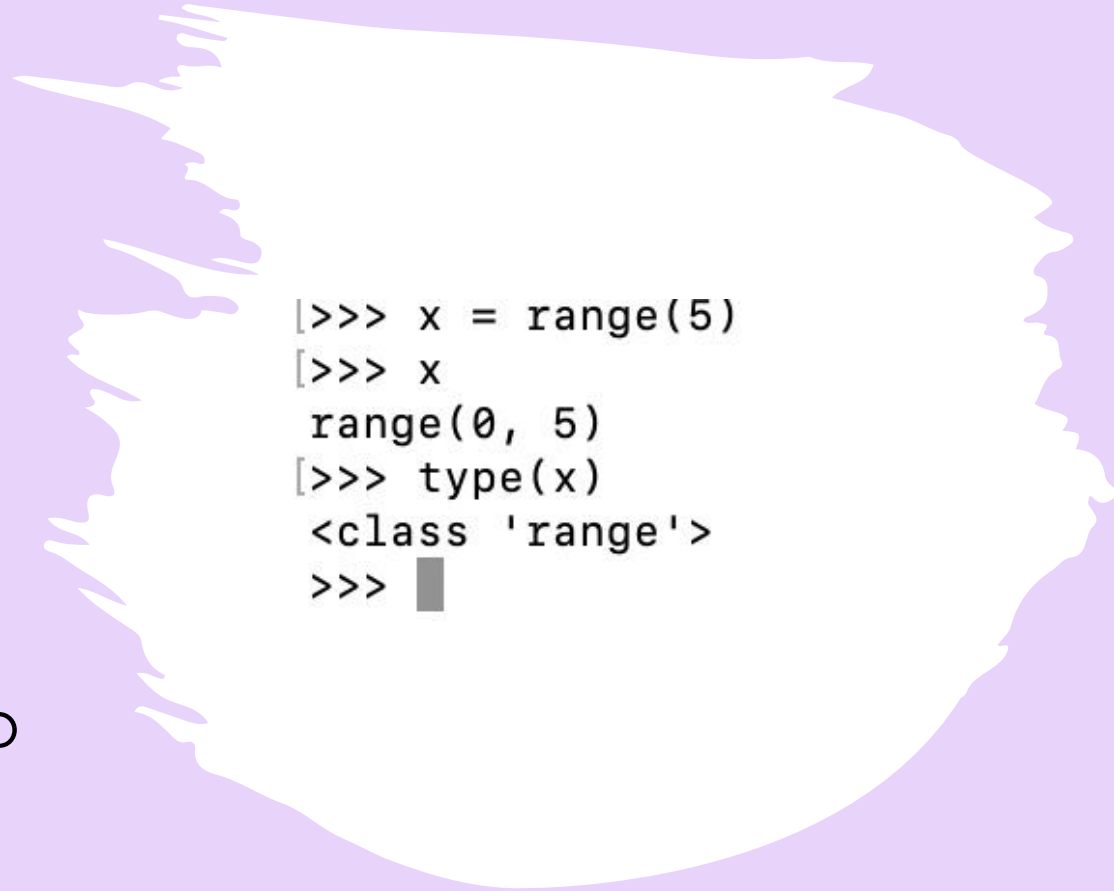
```
def find_max(my_list):
    """
    Returns the maximum integer in the integer list my_list.
    Note: my_list remains unchanged.

    my_list: a list of integers with at least 1 element

    Examples:
    find_max([0]) Returns 0
    find_max([4,0,12]) Returns 12
    find_max([-4,-10,-2]) Returns -2
    """
```

Example: Regular For Loop

LET'S CODE THIS FUNCTION TOGETHER!

Then, How Do We "Loop Over" Integers?

- Answer: range!
- What does range do?
  - Takes an integer and makes it into something of type range.
- What is something of type range?
  - Basically, range(0,5) lets Python assign the values 0, 1, 2, 3, and 4 to a loop variable x.



Range in Action!

Range can have 1 argument (an "ending point")

```
>>> for num in range(5):
...     print(num)
...
0
1
2
3
4
```

Range can have 2 arguments (a "starting" and "ending point"

```
>>> for num in range(0,5):
...     print(num)
...
0
1
2
3
4
```

Range in Action!

Range can also have 3 arguments (a "starting point," "ending point," and "step/what to increment by"

```
>>> for num in range(4,10,2):
...     print(num)
...
4
6
8
```

*** In the above, we want to print the even integers in [4,10). Notice, 4 is the "starting point," 10 is the "ending point (non-inclusive)," and 2 is the "step/what we increment by"**

Wait, Can't We Use Range to Get Indices?

- Yes, you can!
- Say, the_list was of length 6.
- Then, len(the_list) == 6
- Also, range(6) allows Python to iterate through 0, 1, 2, 3, 4, and 5.
- These numbers are the indices of the_list!
- So, range(len(the_list)) allows Python to loop over indices of the_list.

```
>>> the_list = ["loops", "are", "fun", "but", "not", "easy"]
>>> for idx in range(len(the_list)):
...     print(idx)
...
0
1
2
3
4
5
```

```
def clamp(alist, floor, ceiling):
    """
    Does not return anything.  Rather, modifies `alist` so that every element is
    between `floor` and `ceiling`, as follows:

        * Any element less than `floor` is replaced with `floor`.
        * Any element greater than `ceiling` is replaced with `ceiling`.
        * No other elements are changed.

    Example: if `alist` is [-1, 1, 3, 5], then clamp(alist,0,4) changes
    `alist` to have [0,1,3,4] as its contents.

    Preconditions:
        `alist` is a (possibly empty) list of ints
        `floor` and `ceiling are floats or ints, and `floor` <= `ceiling`
    """
```

Example: Regular For Loop V2

LET'S CODE THIS FUNCTION TOGETHER!

Takeaways

Range Loops

- Good when you want to loop through something that isn't a list (like looping through certain integers)
- Best way to get indices!
- **Almost always necessary when you want to change a list**

Regular For Loops

- Good to use when we just need to look at elements of a list or string (or other iterable object)
- **Almost never can be used to edit a list**

```
def skipmult(alist,n,k):
    """MODIFIES alist to multiply every kth element by n (starting at 0).

    Example: Suppose a = [1, 3, 5, 7]. skipmult(a,2,2) makes a == [2, 3, 10, 7]
    (positions 0 and 2). Similarly, skipmult(a,2,3) makes a == [2, 3, 5, 14].

    Precondition: alist is a list of numbers (int or float)
    Precondition: n is number (int or float), k is an int > 1
    """
```

Third Example: Range

LET'S LOOK AT A WAY TO CODE THIS WRONG FIRST

Now You Try!

```
def skipmult(alist,n,k):
    """MODIFIES alist to multiply every kth element by n (starting at 0).

    Example: Suppose a = [1, 3, 5, 7]. skipmult(a,2,2) makes a == [2, 3, 10, 7]
    (positions 0 and 2). Similarly, skipmult(a,2,3) makes a == [2, 3, 5, 14].

    Precondition: alist is a list of numbers (int or float)
    Precondition: n is number (int or float), k is an int > 1
    """
```

Third Example: Range

LET'S SOLVE IT TOGETHER

```
def nullify_dups(table):
    """ This function will edit a table to set all duplicate values to
    0. If an entry in table is listed prior in table, the entry is set to
    0. On an element's first occurrence, it says its original value.

    Recall, a nested list in Python can be thought of as a table, where
    the list [[1,2,3],[1,2,4],[4,5,3]] can be written as the table
        1 2 3
        1 2 4
        4 5 3
    If we called nullify_dups([[1,2,3],[1,2,4],[4,5,3]]), the output would be
        1 2 3
        0 0 4
        0 5 0
    If any entries in the list are 0, they will stay 0.

    Precondition: table is a nested list, where each list inside of table
    is a list of integers
    """
```

Fourth Example: Nested Loops

THIS EXAMPLE REQUIRES NESTED LOOPS; TRY IT OUT!

Now You Try!

```
def nullify_dups(table):
    """ This function will edit a table to set all duplicate values to
    0. If an entry in table is listed prior in table, the entry is set to
    0. On an element's first occurrence, it says its original value.

    Recall, a nested list in Python can be thought of as a table, where
    the list [[1,2,3],[1,2,4],[4,5,3]] can be written as the table
        1 2 3
        1 2 4
        4 5 3
    If we called nullify_dups([[1,2,3],[1,2,4],[4,5,3]]), the output would be
        1 2 3
        0 0 4
        0 5 0
    If any entries in the list are 0, they will stay 0.

    Precondition: table is a nested list, where each list inside of table
    is a list of integers
    """
```

Fourth Example: Nested Loops

LET'S DO IT TOGETHER

Dictionaries

Like Fancy Lists

A few need-to-know things:

To Make A Dictionary

```
>>> grades = {"A": 345, "B": 421, "C": 98, "D": 32}
```

NOTICE: Curly Brackets

Add to an Existing Dictionary

```
>>> grades
{'A': 350, 'B': 421, 'C': 98, 'D': 32}
>>> grades["F"] = 6
>>> grades
{'A': 350, 'B': 421, 'C': 98, 'D': 32, 'F': 6}
```

Edit an Existing Dictionary

```
>>> grades["A"] = grades["A"] + 5
>>> grades
{'A': 350, 'B': 421, 'C': 98, 'D': 32}
```

Empty Dictionary

```
>>> empty = {}
```

Get Elements of a Dictionary

```
>>> grades["A"]
350
```

```
>>> grades["B"]
421
```

```
>>> grades["C"]
98
```

```
def encode(cipher,text):
    """Returns an encoded copy of text using given cipher dictionary
    Example: encode({'a':'o','z':'b'},'razzle') returns 'robble'
    Precondition: cipher is good with lowercase letters as keys and values
    Precondition: text is a (possibly empty) string of lowercase letters"""
```

Last Example: Dictionaries

FIRST, LET'S SHOW HOW TO LOOP OVER DICTIONARIES

Now You Try!

```
def encode(cipher,text):
    """Returns an encoded copy of text using given cipher dictionary
    Example: encode({'a':'o','z':'b'},'razzle') returns 'robble'
    Precondition: cipher is good with lowercase letters as keys and values
    Precondition: text is a (possibly empty) string of lowercase letters"""
```

Last Example: Dictionaries

LET'S DO IT TOGETHER

Thank you for coming!

Any Questions?

Photo Cites

- pbj-today-170427-tease.jpg
- 16687660.jpg