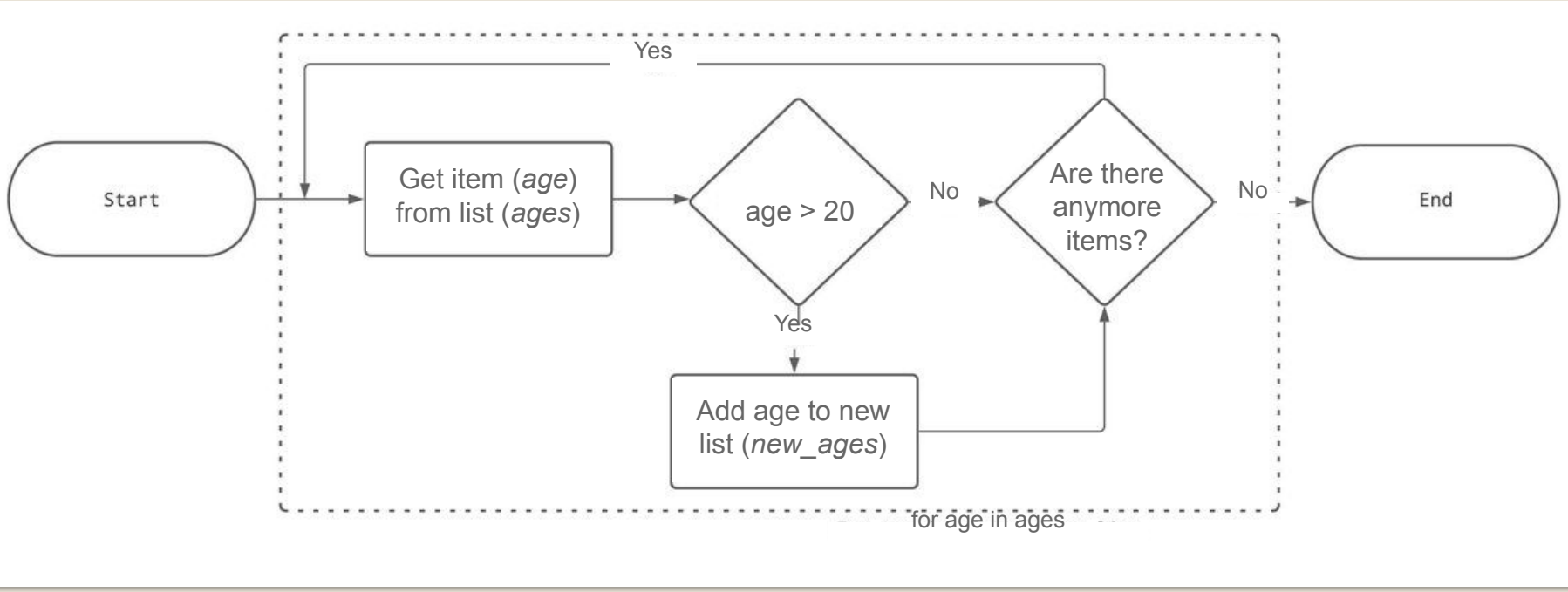
- Building a list
  - List comprehension
  - Identifying the most and least common items
- Defining a function
  - Adding parameters/arguments
  - Keyword parameters

Building a list

For Loop: Building a list

- To create a new list, we need to first create a variable that is an empty list
- What are we sorting/saving into the new list?
  - Conditionals that we will need (if/else)
- Recall the list method *.append*. We will use it to add items to the empty list.
- We can also check the new list length with the function *len().*


Start
Get item (*age*) from list (*ages*)
age > 20
Yes
Add age to new list (*new_ages*)
No
Are there anymore items?
Yes
No
End
for age in ages

Practice: Who’s younger or older than me?

Choose and build a new list of ages for ages that are either older or younger than you. You will need:

- An empty list variable
- Conditional statement to sort those younger or older than you
- Append the sorted ages to the empty list
- Check length of new list and print new list

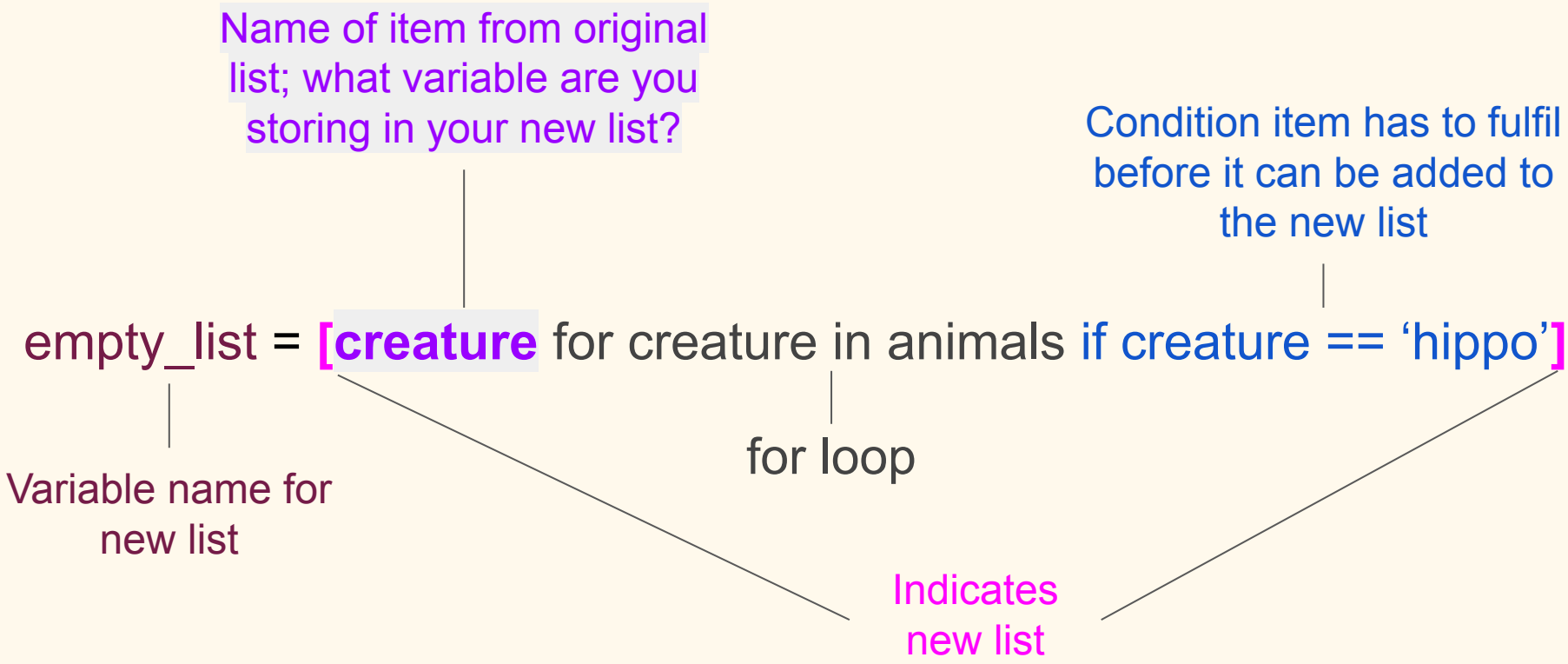
Practice: Who’s younger or older than me?

Choose and build a new list of ages for ages that are either older or younger than you. You will need:

- An empty list variable
  - new_ages = []
- Conditional statement to sort those younger or older than you
  - if age > 20
- Append the sorted ages to the empty list
  - new_ages.append(age)
- Check length of new list and print new list
  - len(new_ages)
  - print(new_ages)

List comprehensions

- Compact way of building a new list
  - One condition:
    - empty_list = [creature for creature in creatures if creature == “hippo”]


Name of item from original list; what variable are you storing in your new list?
Condition item has to fulfil before it can be added to the new list
empty_list = [creature for creature in animals if creature == 'hippo']
Variable name for new list
for loop
Indicates new list

List comprehensions

- Translate the for loop you created in *Who's younger or older than me?*

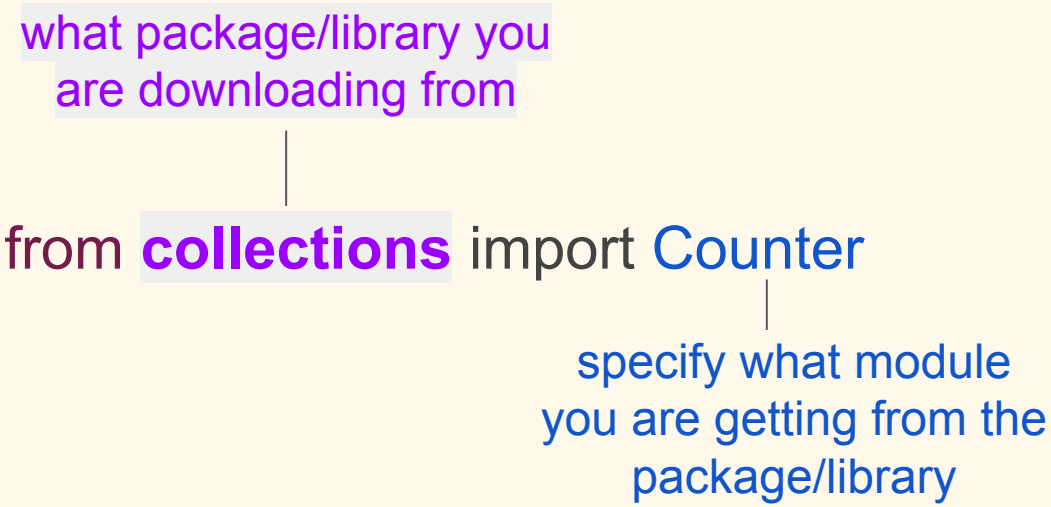
List comprehensions

- More than one condition:
  - empty_list = [creature for creature in creatures if creature == “hippo” or creature == “whale”]

Counting items

Counting items

- Import module, **Counter**, from package collections



Most common items

- Count how many times an item appears (frequency)
  - Counter(ages)
    - Output is another data type called a *dictionary*
- Create new variable with counter
  - ages_tally = Counter(ages)
- Display items from most common to least common
  - ages_tally.most_common()
    - ages_tally.most_common(3) → lists top 3 common items

Most common items

- Display least common item by slicing the list of most_common() from the back
  - ages.most_common()[-1:] → least common item
  - ages.most_common()[-3:] → 3 least common items

Defining a function

```
def <function_name>():
    <code for python to
    perform something>
    return
```

def function to define/create your function

Name of the function you are creating

Don't forget the parentheses

```
def <function_name>():
    <code for python to perform something>
    return
```

Don't forget the colon (:)

Write some code that you want your function to perform

Complete the function with a return statement

```
def happy_birthday():
    print("Happy Birthday to you")
    print("Happy Birthday to you")
    print("Happy Birthday dear human life form")
    print("Happy Birthday to you")
    return
```

Practice: Defining a function

Make a function that prints your favorite greeting! You will need to begin with *def* and a name for your function.

- **def** <function_name>()**:**
  <code for python to perform something>
  return

Adding parameters/argument

Allows for values to be added to your function; can be named anything (like a variable name)

```
def <function_name>(<parameter>):
    <code for python to
    perform something>
    return
```

Parameters and arguments

- parameter = human (thing that requires a value for the function)
- argument = “Di” (actual value passed to function)

```
def personalized_happy_birthday(human):
    print("Happy Birthday to you")
    print("Happy Birthday to you")
    print(f"Happy Birthday {human}")
    print("Happy Birthday to you")
    return
```

Practice: Adding a parameter

Add a parameter to your greeting function for a user to add their name to the greeting.

- **def** <function_name>(<parameter>)**:**
  <code for python to perform something>
  return

Keyword arguments

Keyword argument allows for explicit definition of values

arg can be assigned a default value

```
def <function_name>(<parameter_name> = arg):
    <code for python to perform something>
    return
```

Keyword arguments

- Explicitly define your arguments with keyword arguments
- Useful when defining multiple parameters
- Use an = sign to assign default values

```
def keyword_happy_birthday(to_name, from_name = me):
    print("Happy Birthday to you")
    print("Happy Birthday to you")
    print(f"Happy Birthday {to_name}")
    print("Happy Birthday to you")
    print(f"\nSincerely, \n{from_name}")
    return
```

Practice: Adding keyword arguments

Add at least 2 keyword arguments to your greeting function that defines default values for the greeting.

- **def** <function_name>(<parameter> = “human life form”)**:**
  <code for python to perform something>
  return

Returning a specific value

```
def calculate_dog_years_age(age):
    dog_years_age = age * 7
    return dog_years_age
```

Specify the value for
the function to return