

Midterm Review

Rob Thompson

UW CSE 160

Winter 2021

Midterm Setup

- Releases 8:00pm tonight
- Due 11:00pm Wednesday
- It will not take you 51 hours to finish
- Open-book, open-note, open-lecture, open-class-website
- Can ask questions on Ed and in Office Hours
- Can share anything within your group

ARE WE REALLY READY FOR THIS?

Yes! You've learned a lot, and you've been building a knowledge base this whole time.

You type *expressions*. Python computes their *values*.

- 5
- 3 + 4
- 44 / 2
- 2 ** 3
- 3 * 4 + 5 * 6
 - If precedence is unclear, use parentheses
- (72 – 32) / 9.0 * 5

Variables hold values

- Recall variables from algebra:
 - Let $x = 2$...
 - Let $y = x$...
- In Python assign a variable: “*varname = expression*”

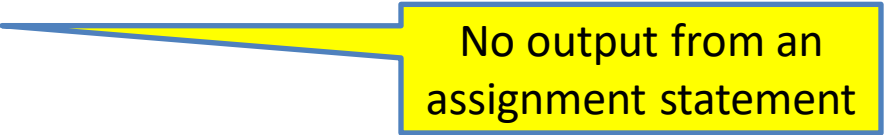
```
pi = 3.14
```

```
pi
```

```
avogadro = 6 * 10 ** 23
```

```
avogadro
```

```
22 = x          # Error!
```



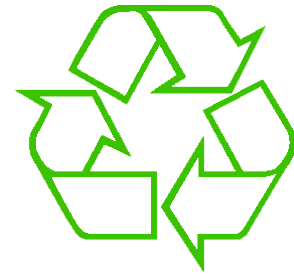
No output from an assignment statement

- Not all variable names are permitted

Types of values

- Integers (**int**): -22, 0, 44
 - Arithmetic is **exact**
- Real numbers (**float**) : 2.718, 3.1415
 - **float**, for “floating point”
 - Arithmetic is **approximate**
- Strings (**str**): "I love Python", ""
- Truth values (**bool**): True, False
 - **bool**, for “Boolean”

for Loop Explained



[See in python tutor](#)

A better way to repeat yourself:

```
for fahr in [30, 40, 50, 60, 70]:  
    cent = (fahr - 32) / 9.0 * 5  
    print(fahr, cent)  
print("All done")
```

Output:
30 -1.11
40 4.44
50 10.0
60 15.56
70 21.11
All done

The range function

A typical for loop does not use an explicit list:

```
for i in range (5) :
```

```
... body ...
```

Upper limit
(*exclusive*)

Produces a range
object

range (5) → will loop through [0, 1, 2, 3, 4]

Lower limit
(*inclusive*)

range (1, 5) → will loop through [1, 2, 3, 4]

step (distance
between elements)

range (1, 10, 2) → will loop through [1, 3, 5, 7, 9]

Nested Loops

```
for i in [1, 2, 3]:  
    print("Before j loop i is", i)  
    for j in [50, 100]:  
        print("j is", j)
```

What is the output?

Using If to find absolute value

If the value is negative, negate it.

Otherwise, use the original value.

```
val = -10

# calculate absolute value of val
if val < 0:
    result = -val
else:
    result = val

print(result)
```

Condition must be a Boolean expression

Indentation is significant

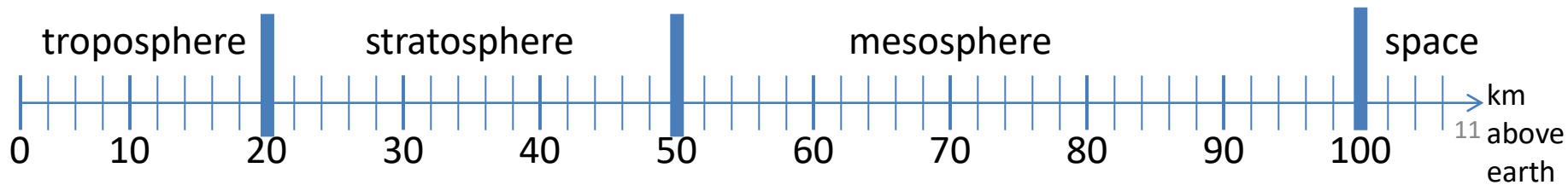
else is not required

In this example, **result** will always be assigned a value.

Version 3 (Best)

```
if height > 100:  
    print("space")  
elif height > 50:  
    print("mesosphere")  
elif height > 20:  
    print("stratosphere")  
else:  
    print("troposphere")
```

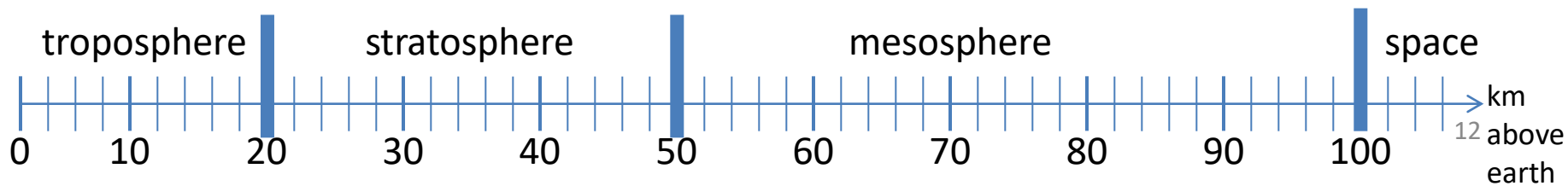
ONE of the print statements is guaranteed to execute:
whichever condition it encounters first that is true



What Happens Here? (bad example)

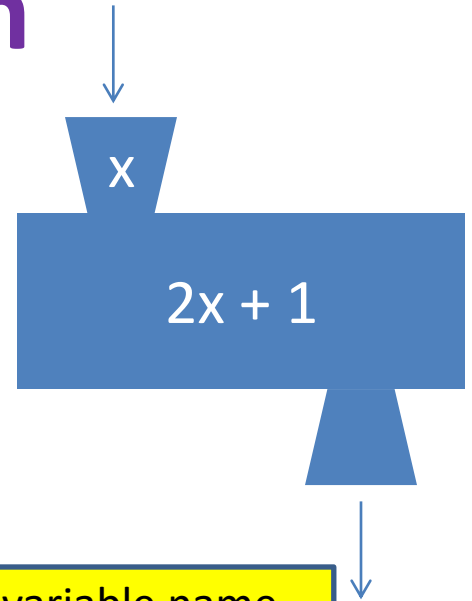
```
# height is in km
if height > 100:
    print("space")
if height > 50:
    print("mesosphere")
if height > 20:
    print("stratosphere")
else:
    print("troposphere")
```

Try height = 72



Creating a function

Define the machine,
including the input and the result



Name of the function.
Like “y = 5” for a variable

Keyword that means:
I am **def**ining a function

Input variable name,
or “formal parameter”

```
def dbl_plus (x) :
```

```
return 2 * x + 1
```

Keyword that means:
This is the result

Return expression
(part of the **return** statement)

How to look up a variable

Idea: find the nearest variable of the given name

1. Check whether the variable is defined in the **local scope**
2. ... check any intermediate scopes (**none** in CSE 160!) ...
3. Check whether the variable is defined in the **global scope**

If a local and a global variable have the **same name**, the global variable is inaccessible (“**shadowed**”)

This is confusing; try to avoid such shadowing

```
x = 22
stored = 100
def lookup():
    x = 42
    return stored + x
val = lookup()
x = 5
stored = 200
val = lookup()
```

Local variables exist only while the function is executing

```
def cent_to_fahr(cent):  
    result = cent * 5.0 * 9 + 32  
    return result
```

[See in python tutor](#)

```
tempf = cent_to_fahr(5)  
print(result)
```

ALL of your variables will be local for the midterm

How to design a function

1. **Wishful thinking:**

Write the program as if the function already exists

2. Write a **specification:**
Describe the inputs and output, including their types

No implementation yet!

3. Write **tests:** Example inputs and outputs

4. Write the function **body** (the implementation)

First, write your plan in English, then translate to Python

```
def fahr_to_cent(fahr):  
    """Input: a number representing degrees Farenheit  
    Return: a number representing degrees centigrade  
    """  
    res = (fahr - 32) * 5 / 9  
    return res
```

Not relevant for the midterm

Already done for you!

We did *some* of these for you, you should make more to be sure

This is the main part of the midterm

What is a list?

- A list is an ordered sequence of values

- A list of integers:

[3, 1, 4, 4, 5, 9]

0	1	2	3	4	5
3	1	4	4	5	9

- A list of strings:

["Four", "score", "and", "seven", "years"]

0	1	2	3	4
"Four"	"score"	"and"	"seven"	"years"

- Each value has an **index**
 - Indexing is zero-based (counting starts with zero)
- `len([3, 1, 4, 4, 5, 9])` returns 6

List Creation

```
a = [ 3, 1, 2 * 2, 1, 10 / 2, 10 - 1 ]
```

3	1	4	1	5	9
---	---	---	---	---	---

```
b = [ 5, 3.0, 'hi' ]
```

```
c = [ 4, 'a', a ]
```

```
d = [ [1, 2], [3, 4], [5, 6] ]
```

List Querying

0	1	2	3	4	5
3	1	4	4	5	9

Expressions that return parts of lists:

- Single element: `mylist[index]`
 - The single element stored at that location
- Sublist (“slicing”): `mylist[start:end]`
 - the sublist that starts at index `start` and ends at index `end - 1`
 - If `start` is omitted: defaults to 0
 - If `end` is omitted: defaults to `len(mylist)`
 - `mylist[:]` evaluates to the whole list
 - `mylist[0:len(mylist)]` also does

More List Querying

- Find/lookup in a list

`x in mylist`

- Returns True if **`x`** is found in **`mylist`**

`mylist.index(x)`

- Return the integer index in the list of the *first item* whose value is **`x`**.
- It is an error if there is no such item.

`mylist.count(x)`

- Return the number of times **`x`** appears in the list.

0	1	2	3	4	5
3	1	4	4	5	9

List Insertion

0	1	2	3	4	5
3	1	4	4	5	9

- `mylist.append(x)`
 - Extend `mylist` by inserting `x` at the end
- `mylist.extend(L)`
 - Extend `mylist` by appending all the items in the argument list `L` to the end of `mylist`
- `mylist.insert(i, x)`
 - Insert item `x` before position `i`.
 - `a.insert(0, x)` inserts at the front of the list
 - `a.insert(len(a), x)` is equivalent to
`a.append(x)`

Note: `append`, `extend` and `insert` all return `None`

List Removal

- `mylist.remove(x)`
 - Remove the first item from the list whose value is **x**
 - It is an error if there is no such item
 - Returns **None**

Notation from the Python Library Reference:
The square brackets around the parameter, “[i]”, means the argument is *optional*.
It does *not* mean you should type square brackets at that position.

- `mylist.pop([i])`
 - Remove the item at the given position in the list, and return it.
 - If no index is specified, `a.pop()` removes and returns the last item in the list.

Note: `remove` returns `None`

List Replacement

- `mylist[index] = newvalue`
- `mylist[start:end] = newsublist`
 - Replaces `mylist[start]... mylist[end - 1]` with `newsublist`
 - Can change the length of the list
- `mylist[start:end] = []`
 - removes `mylist[start]... mylist[end - 1]`
- `mylist[len(mylist):] = L`
 - is equivalent to `a.extend(L)`

List expression examples

What does this mean (or is it an error)?

```
["four", "score", "and", "seven", "years"][2]
```

```
["four", "score", "and", "seven", "years"][0,2,3]
```

```
["four", "score", "and", "seven", "years"][[0,2,3]]
```

```
["four", "score", "and", "seven", "years"][[0,2,3][1]]
```


Reading a file in python

```
# Open takes a filename and returns a file object.  
# This fails if the file cannot be found & opened.  
myfile = open("datafile.dat")
```

```
# Approach 1: Process one line at a time  
for line_of_text in myfile:  
    ... process line_of_text
```

```
# Approach 2: Process entire file at once  
all_data_as_a_big_string = myfile.read()
```

```
myfile.close() # close the file when done reading
```

Assumption: file is a sequence of lines

Where does Python expect to find this file (note the relative pathname)?

Writing to a file in python

Replaces any existing file of this name

```
myfile = open("output.dat", "w")
```

open for **Writing**
(no argument, or
"r", for Reading)

Just like printing output

```
myfile.write("a bunch of data")
```

```
myfile.write("a line of text\n")
```

"\n" means
end of line
(Newline)

```
myfile.write(4)
```

Incorrect; results in:

TypeError: expected a character buffer object

```
myfile.write(str(4))
```

Correct. Argument
must be a string

```
myfile.close()
```

close when done
with all writing