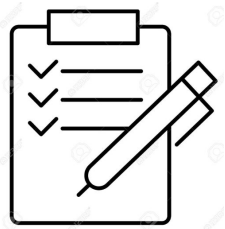
Intro to Programming

3-12-19

Sign In!

http://tinyurl.com/CADIntroPySp19-4

Last Week's Review



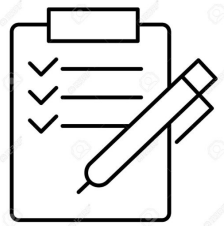
We have a boolean `rain` that tells us if it will rain tomorrow and a boolean `test` that tells us if there's a test tomorrow. We also have a list `schedule = ["Wake up", "Go to class", "Sleep"]`.

We want to add `"Bring your umbrella!"` to our schedule before `"Go to class"` if it's raining. We also want to add `"Study for test!"` before `"Sleep"` if there's a test, or both if there's a test and it's raining. If neither are true, you can just print out your current schedule.

Write a program that does this.

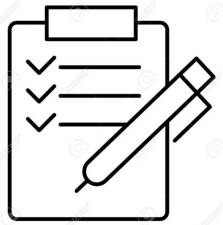
Last Week's Review Answer

```
schedule = ["Wake up", "Go to class", "Sleep"]

if rain == True and test == True:
    schedule.insert(1, "Bring your umbrella!")
    schedule.insert(3, "Study for test!")
elif rain == True:
    schedule.insert(1, "Bring your umbrella!")
elif test == True:
    schedule.insert(-1, "Study for test!")
else:
    print(schedule)
```



Last Week's Review (Better) Answer

```
schedule = ["Wake up", "Go to class", "Sleep"]

if not rain and not test:
    print(schedule)
if rain:
    schedule.insert(1, "Bring your umbrella!")
if test:
    schedule.insert(-1, "Study for test!")
```



Useful List Methods - len()

- `len()` - this returns the length of the list (or a string!)

```
teachers = [“Lin”, “Gheith”, “Scott”, “Norman”, “Young”]
num_teachers = len(teachers)
print(num_teachers)
print(len(teachers[0]))
```

OUTPUT:

```
5
3
```

Useful List Methods - sort()

- `sort()` - this sorts list in alphabetical or numeric order

```
teachers = [“Lin”, “Gheith”, “Scott”, “Norman”, “Young”]
teachers.sort()
print(teachers)
```

OUTPUT: [“Gheith”, “Lin”, “Norman”, “Scott”, “Young”]

List Slicing

- You can keep a subpart of a list by **splicing** it.
- We use the `[first:last]` structure to do this.
- The resulting sublist will be inclusive of the first index and exclusive of the last index.
- This does NOT change the original list.
- EX: alphabet[1:4]

index	0	1	2	3	4
element	a	b	c	d	e

List Slicing

```
my_schedule =  ["Jazz",
"Basket Weaving", "Japanese",
"Intro to Python"]

l = len(my_schedule)
```

```
AM_classes = my_schedule[0 : l/2]

PM_classes = my_schedule[l/2 : l]
```

List Slicing

```
my_schedule =  ["Jazz",
"Basket Weaving", "Japanese",
"Intro to Python"]

l = len(my_schedule) = 4
```

my_schedule:

0	1	2	3
Jazz	Basket Weaving	Japanese	Intro to Python

```
AM_classes = my_schedule[0 : 2]

PM_classes = my_schedule[2 : 4]
```

List Slicing

```
my_schedule =  ["Jazz",
"Basket Weaving", "Japanese",
"Intro to Python"]

l = len(my_schedule) = 4
```

my_schedule:

0	1	2	3
Jazz	Basket Weaving	Japanese	Intro to Python

```
AM_classes = my_schedule[0 : 2]

PM_classes = my_schedule[2 : 4]
```

AM_classes:

0	1
Jazz	Basket Weaving

PM_classes:

0	1
Japanese	Intro to Python

Loops

- Oftentimes we'll want to access all entries in a list, performing the same task with each item
- Indexing the same list and over again is inefficient and tedious

```
instructors = [“Sean”, “Claire”, “Hannah”]
message = “ is an instructor.”
print(instructors[0] + message)
print(instructors[1] + message)
print(instructors[2] + message)
```

Output

```
Sean is an instructor.
Claire is an instructor.
Hannah is an instructor.
```

Loops Cont'd

- A basic for loop uses the syntax:

```
for variable in iterable:
        # your code here
```

- The `for` keyword tells Python to read a value from the `iterable` and assign it to `variable`. Python will then repeat those two lines of code until it reaches the end of the `iterable`.
- Loops visualized: https://goo.gl/45bh64

```
instructors = ["Sean", "Claire", "Hannah"]
message = " is an instructor."

for instructor in instructors:
    print(instructor + message)
```

Output

```
Sean is an instructor.
Claire is an instructor.
Hannah is an instructor.
```

Loops Cont'd

- A basic for loop uses the syntax:

```
for variable in list:
    # your code here
```

- The `for` keyword tells Python to read a value from the `list` and assign it to `variable`. Python will then repeat those two lines of code until it reaches the end of the `list`.
- Loops visualized: https://goo.gl/45bh64

```
instructors = ["Sean", "Claire", "Hannah"]
message = " is an instructor."

for instructor in instructors:
    print(instructor + message)
```

Output

```
Sean is an instructor.
Claire is an instructor.
Hannah is an instructor.
```

Common mistakes to avoid

- Python uses indentation to determine if a line of code is connected to the one above it
- As you write code that relies on indented blocks, watch out for indentation errors.
- Always indent the line after the `for` statement in a loop

```
instructors = ["Sean", "Claire",
"Hannah"]
message = " is an instructor."

for instructor in instructors:
print(instructor + message)
```

Output:

```
    print(instructor + message)
        ^
IndentationError: expected an
indented block
```

Mistakes Cont'd

- Don't forget the colon at the end of a `for` statement
- The following code would result in an error:

```
for instructor in instructors
    print(instructor + message)
```

- Indenting unnecessarily can cause unexpected behavior:
- For example, say we wanted to print a thank you message after listing out all of the instructors.
- The following code would print it too many times:

```
for instructor in instructors
    print(instructor + message)
    print("Thank you everyone for
    teaching this workshop!")
```

Output:

```
Sean is an instructor.
Thank you everyone for teaching this
workshop!
Claire is an instructor.
Thank you everyone for teaching this
workshop!
Hannah is an instructor.
Thank you everyone for teaching this
workshop!
```

Making numerical lists

- Python's range() function makes it easy to generate a series of numbers.
- Note that the example only prints numbers 1-4 instead of 1-5.
- The range function has the syntax `range(start, end)` where end is exclusive.
- If no `start` value is given it will default to 0.
- Examples:
  - range(6) -> [0,1,2,3,4,5]
  - range(2,7) -> [2,3,4,5,6]

```
for value in range(1,5):
    print(value)
```

Output:

1

2

3

4

Coding exercise:

- Using a for loop and the range function, calculate the first 10 `square` numbers and store them into a list.
- `squares` should read
  - [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Coding exercise:

- Using a for loop and the range function, calculate the first 10 square numbers and store them into a list.
- Starting from the code we've written together, modify it so that `squares` now only stores odd values. If a square is even, print the base along with a message:
  - EX: "The square of 2 is even"!
  -

Hints:

- Use an if/else statement
- Use the % operator (remainder division)
  - EX: 5 % 5 = 0
    (5 divides evenly into 5. There is no remainder, so the result is 0)
  - EX: 5 % 2 = 1
    (2 doesn't divide evenly into 5. As such, there is a remainder of one.)
  - By doing % 2 on any number we can check if it is odd or even
    - odd % 2 = 1, even % 2 = 0

Thanks for coming!

- Next week - Methods
- Please fill out our feedback form, especially if you'd like to specify which topics we cover next week!
    - http://tinyurl.com/CADIntroPyFeedback4