Announcements

- Homework
  - Homework 2 has been posted!
  - I may well still be working on getting through grading all your HW1. Hopefully by the end of today all feedback will be posted
  - If you are still working on HW1, try to get it in soon! Late days/hours are burning away!
- Remember that tutors are available Monday and Thursday nights, from 7-10pm
- Polling: `rembold-class.ddns.net`

Review Question

What will the output of the below code be upon completion?

```
x = 7
y = 0
while x >= 0:
    y = y + x
    x = x - 2
print(y)
```

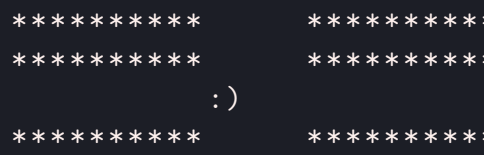
A) 15

B) 16

C) 28

D) Code will not complete (infinite loop)

Lost Woods Examples



Nesting Loops

- You are allowed to nest loops, and in many cases might want to
- Realize that one loop must finish before the next advances one iteration!
  - At which point the inner loop would run in its entirety again

```
week = 1
day = 1

while week <= 52:
    while day <= 3:
        if day == 1:
            print('Monday')
        elif day == 2:
            print('Wednesday')
        elif day == 3:
            print('Friday')

        day = day + 1
    week = week + 1
```

Taking a break

- Sometimes it can be useful to exit a loop without checking the initial test
  - Maybe you have multiple conditions that could stop the loop
  - Maybe you have a special condition that when certain requirements are met the loop ends
- The `break` statement will force an immediate exit from whatever loop it happens to be in

```
x = 1
while True:
    print(x)
    x = x + 1
    if x > 10:
        break
```

Understanding Check

What would be the *last* number printed by the below code?

```
x = 1
while x < 10:
    y = 10
    while y > x:
        if y % 3 == 0:
            break
        print(x*y)
        y = y - 2
    x = x + 1
```

A) 90
B) 60
C) 56
D) 25

Dissecting Strings

- Recall that strings are a non-scalar object
  - Made up of smaller pieces, characters in the case of strings
- Each character has an "address" within the string where it lives, called its index.

```
"Spaghetti"
```

Dissecting Strings

- Recall that strings are a non-scalar object
  - Made up of smaller pieces, characters in the case of strings
- Each character has an "address" within the string where it lives, called its index.

```
"Spaghetti"
 1 2 3 4 5 6 7 8 9
```

Dissecting Strings

- Recall that strings are a non-scalar object
  - Made up of smaller pieces, characters in the case of strings
- Each character has an "address" within the string where it lives, called its index.

"Spaghetti"

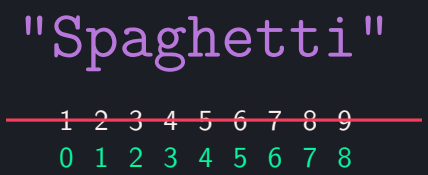
1 2 3 4 5 6 7 8 9

Start at 0!

Counting (and indexing) in Python always starts with 0!! This is easy to forget!

Dissecting Strings

- Recall that strings are a non-scalar object
  - Made up of smaller pieces, characters in the case of strings
- Each character has an "address" within the string where it lives, called its index.

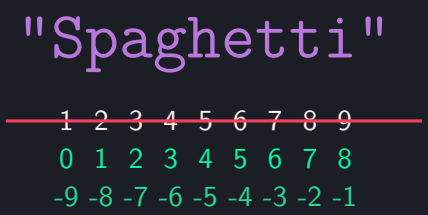


Start at 0!

Counting (and indexing) in Python always starts with 0!! This is easy to forget!

Dissecting Strings

- Recall that strings are a non-scalar object
  - Made up of smaller pieces, characters in the case of strings
- Each character has an "address" within the string where it lives, called its index.



Start at 0!

Counting (and indexing) in Python always starts with 0!! This is easy to forget!

Indexing Notation

- To get the character at a certain address or index, use *square brackets*

```
x = "Spaghetti"
```

```
0 1 2 3 4 5 6 7 8
```

- `x[0] = "S"`
- `x[4] = "h"`
- `x[-1] = "i"`
- `x[1] = "p"`

The Slicing

- Frequently, you want more than a single character
- Would be useful to give a start and stop point to extract a "sub"-string
  - Possible through slicing!
  - Notation is: `var[<start>:<stop>]`
  - Could be read as: "Get me all the characters from the `<start>` up to the `<stop>` *but not including the stop*"

```
x = "Spaghetti"

    0 1 2 3 4 5 6 7 8
```

  - `x[0:3] = "Spa"`
  - `x[4:] = "hetti"`

and the Dicing

- Slices are a form of shorthand for looping through a string and pulling out the value between two indices
- But what if you don't want *every* character between the two endpoints?
  - Only want every other character? Or every 10th character?
- Can achieve by adding one more term to our slices:

`x[<start>:<stop>:<step>]`

  - A `<step>` of 2 would take every other character
  - Default is a `<step>` of 1
  - A *negative* step will go backwards from stop to start!