

BRANCHING, ITERATION

(download slides and .py files 'follow along!')

6.0001 LECTURE 2

LAST TIME

- syntax and semantics
- scalar objects
- simple operations
- expressions, variables and values

TODAY

- string object type
- branching and conditionals
- indentation
- iteration and loops

STRINGS

- letters, special characters, spaces, digits
- enclose in **quotation marks or single quotes**

```
hi = "hello there"
```

- **concatenate strings**

```
name = "ana"
```

```
greet = hi + name
```

```
greeting = hi + " " + name
```

- do some **operations** on a string as defined in Python docs

```
silly = hi + " " + name * 3
```

INPUT/OUTPUT: print

- used to **output** stuff to console
- keyword is `print`

```
x = 1  
print(x)  
  
x_str = str(x)  
print("my fav num is", x, ".", "x =", x)  
print("my fav num is " + x_str + ". " + "x = " + x_str)
```

INPUT/OUTPUT: `input ("")`

- prints whatever is in the quotes
- user types in something and hits enter
- binds that value to a variable

```
text = input("Type anything... ")  
print(5*text)
```

- `input` **gives you a string** so must cast if working with numbers

```
num = int(input("Type a number... "))  
print(5*num)
```

COMPARISON OPERATORS ON int, float, string

- i and j are variable names *have to be same type*
- comparisons below evaluate to a Boolean

$i > j$

$i \geq j$

$i < j$

$i \leq j$

$i == j \rightarrow \text{equality}$ test, True if i is the same as j

$i != j \rightarrow \text{inequality}$ test, True if i not the same as j

String comparison is lexicographic alphabetical order - comp

LOGIC OPERATORS ON bools

- a and b are variable names (with Boolean values)

not a → True if a is False
False if a is True

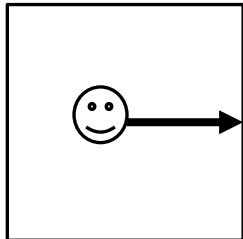
a and b → True if both are True

a or b → True if either or both are True

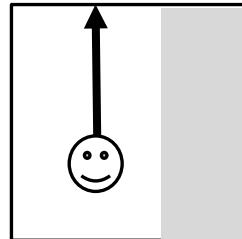
A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

COMPARISON EXAMPLE

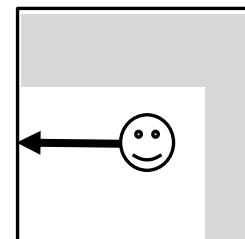
```
pset_time = 15
sleep_time = 8
print(sleep_time > pset_time)
derive = True
drink = False
both = drink and derive
print(both)
```



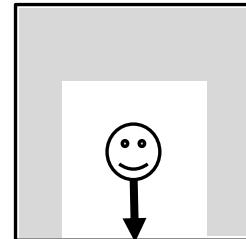
If right clear,
go right



If right blocked,
go forward

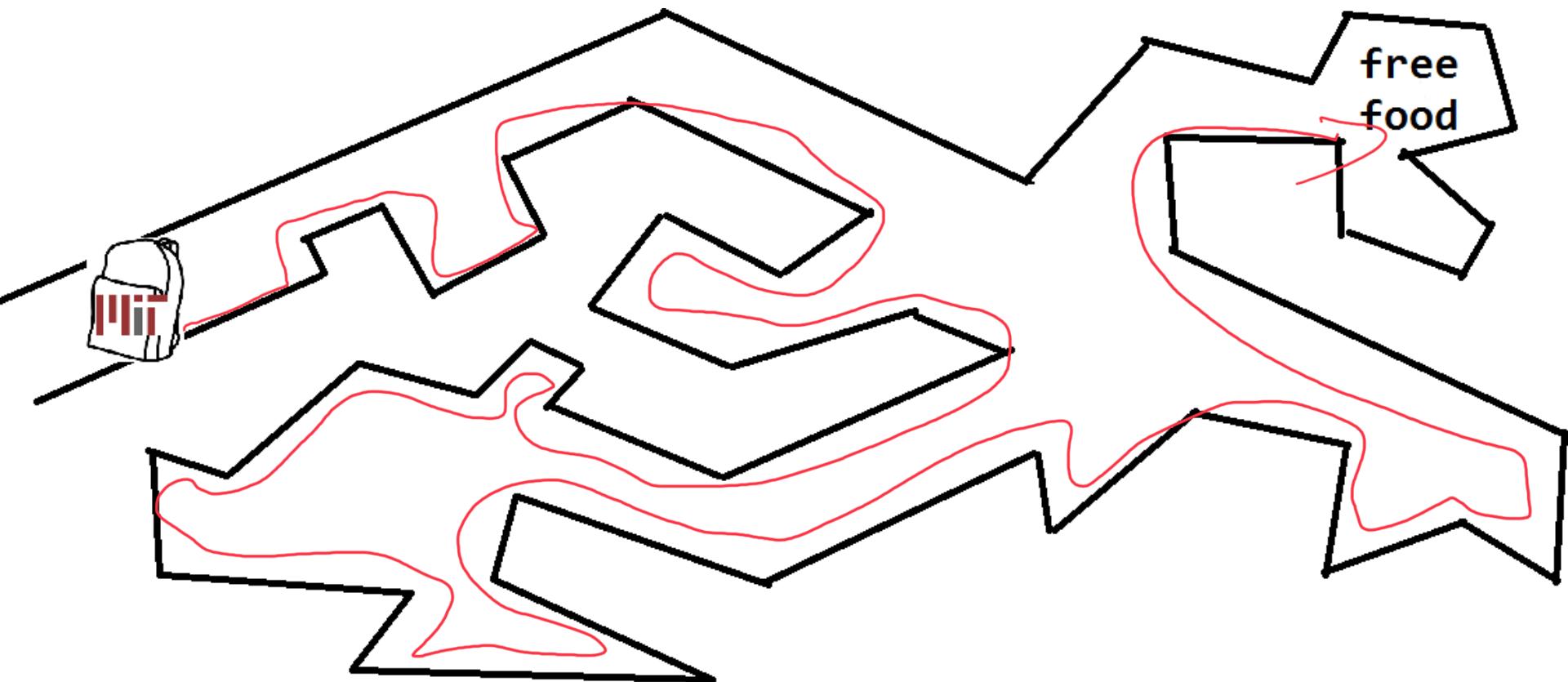


If right and
front blocked,
go left



If right , front,
left blocked,
go back

hand on right wall



CONTROL FLOW - BRANCHING

```
if <condition>:  
    <expression>  
    <expression>  
    ...
```

```
if <condition>:  
    <expression>  
    <expression>  
    ...  
else:  
    <expression>  
    <expression>  
    ...
```

```
if <condition>:  
    <expression>  
    <expression>  
    ...  
elif <condition>:  
    <expression>  
    <expression>  
    ...  
else:  
    <expression>  
    <expression>  
    ...
```

- <condition> has a value True or False
- evaluate expressions in that block if <condition> is True

INDENTATION

typically 4 spaces

- matters in Python
- how you denote blocks of code

```
x = float(input("Enter a number for x: "))
y = float(input("Enter a number for y: "))
if x == y:
    print("x and y are equal")
    if y != 0:
        print("therefore, x / y is", x/y)
elif x < y:
    print("x is smaller")
else:
    print("y is smaller")
print("thanks!")
```

= VS ==

```
x = float(input("Enter a number for x: "))
y = float(input("Enter a number for y: "))
if x == y:
    print("x and y are equal")
    if y != 0:
        print("therefore, x / y is", x/y)
elif x < y:
    print("x is smaller")
else:
    print("y is smaller")
print("thanks!")
```

What if x=y here?
get a SyntaxError



- Legend of Zelda – Lost Woods

- keep going right, takes you back to this same screen, stuck in a loop

Image Courtesy Nintendo, All Rights Reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

```
if <exit right>:  
    <set background to woods_background>  
    if <exit right>:  
        <set background to woods_background>  
        if <exit right>:  
            <set background to woods_background>  
            and so on and on and on...  
        else:  
            <set background to exit_background>  
    else:  
        <set background to exit_background>  
else:  
    <set background to exit_background>
```



- Legend of Zelda – Lost Woods

- keep going right, takes you back to this same screen, stuck in a loop

Word Cloud copyright unknown, All Right Reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

```
while <exit right>:  
    <set background to woods_background>  
<set background to exit_background>
```

CONTROL FLOW: while LOOPS

```
while <condition>:  
    <expression>  
    <expression>  
    ...
```

- <condition> evaluates to a Boolean
- if <condition> is True, do all the steps inside the while code block
- check <condition> again
- repeat until <condition> is False

while LOOP EXAMPLE

You are in the Lost Forest.



Go left or right?

PROGRAM:

```
n = input("You're in the Lost Forest. Go left or right? ")  
while n == "right":  
    n = input("You're in the Lost Forest. Go left or right? ")  
print("You got out of the Lost Forest!")
```

CONTROL FLOW: while and for LOOPS

- iterate through numbers in a sequence

```
# more complicated with while loop
n = 0      counter initialization
while n < 5:    ↪ affects conditions
    print(n)
    n = n+1 incrementing
```

```
# shortcut with for loop
for n in range(5):
    print(n)
```

CONTROL FLOW: for LOOPS

```
for <variable> in range(<some_num>):  
    <expression>  
    <expression>  
    ...
```

general composition of for loop

- each time through the loop, <variable> takes a value
- first time, <variable> starts at the smallest value
- next time, <variable> gets the prev value + 1
- etc.

range (start, stop, step)

- default values are start = 0 and step = 1 and optional
- loop until value is stop - 1

```
mysum = 0
```

```
for i in range(7, 10):  
    mysum += i
```

```
print(mysum)
```

1 arg (stop)

2 arg (start, stop)

3 arg (start, stop, step)

```
mysum = 0
```

```
for i in range(5, 11, 2):  
    mysum += i
```

```
print(mysum)
```

break STATEMENT

- immediately exits whatever loop it is in
- skips remaining expressions in code block
- exits only innermost loop!

```
while <condition_1>:  
    while <condition_2>:  
        <expression_a>  
        break  
        <expression_b>  
    <expression_c>
```

break STATEMENT

```
mysum = 0  
for i in range(5, 11, 2):  
    mysum += i  
    if mysum == 5:  
        break  
    mysum += 1  
print(mysum)
```

- what happens in this program?

mysum = 5

for

VS while LOOPS

for loops

- **know** number of iterations
- can **end early** via break
- uses a **counter** built-in
- **can rewrite** a for loop using a while loop

==

while loops

- **unbounded** number of iterations
- can **end early** via break
- can use a **counter but must initialize** before loop and increment it inside loop
- **may not be able to rewrite** a while loop using a for loop

Like User input

MIT OpenCourseWare
<https://ocw.mit.edu>

6.0001 Introduction to Computer Science and Programming in Python
Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.