Branching & Iteration

Today

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

STRINGS

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

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

- **concatentate** strings

```
name = "andy"

greet = hi + name

greeting = hi + " " + name
```

- **do some operations on a string**

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

INPUT/OUTPUT: `print`

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

```python
x = 1
print(x)
x_str = str(x)
print("my fav num is", x, ".", "x =", x)
print("my fav num is " + x_str + ". " + 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("What is your name?")

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
- comparisons below evaluate to a Boolean

```
i > j
i >= j
i < j
i <= j
i == j # equality test, True if i is the same as j
i != j # inequality test, True if i no the same as j
```

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

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

- matters in Python
- how you denote blocks of code

```python
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!")
```

Legend of Zelda - Lost Woods

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

```
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

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

```
while exit_right:
  # set background to woods_background
else:
  # 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?
```

```
n = input("You are in the Lost Forest\n****************\n****************\n :)\n****************\n****************\nGo left or right? ")

while n == "right" or n == "Right":
    n = input("You are in the Lost Forest\n****************\n******       ***\n  (╯°□°）╯︵ ┻━┻\n****************\n****************\nGo left or right? ")
print("\nYou got out of the Lost Forest!\n\o/")
```

CONTROL FLOW: `while` and `for` LOOPS

- iterate through numbers in a sequence

```
# more complicated with while loop
n = 0

while n < 5:
  print(n)
  n = n + 1

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

CONTROL FLOW: `for` LOOPS

```
for your_variable in range(some_number):
    # expression
    # expression
```

- each time through the loop, your_variable takes a value
- first time, your_variable starts at the smallest value
- next time, your_variable gets the previous value + 1
- etc.

range (start, stop, step)

- default values are `start = 0` and `step = 1` (if omitted)
- loop until value is `stop - 1`

```python
mysym = 0

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

print(mysum)


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

what happens in this program?

```
mysum = 0

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

    if mysum == 5:
        break
        mysum += 1

print(mysum)
```

`for` VS `while` LOOPS

`for` loops

- **know** number of iterations
- can **end early** via break
- uses a **counter**
- **can rewite** 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