1. Recap
○○○○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○○○

KOLT Python

Branching, While Loops, Turtle Graphics & Strings

Necla Mutlu

Tuesday 11th February, 2020

KOÇ UNIVERSITY

OFFICE OF LEARNING AND TEACHING

1. Recap
ooooo

2. Strings
oooo

3. While Loops
oo

4. Turtle
ooooooo

Agenda

1. Recap

2. Strings

3. While Loops

4. Turtle

1. Recap
●○○○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○○○○

Branching

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

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

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

- <condition> has a **bool** value (True or False)
- Which expressions will be evaluated in which conditions?

1. Recap
○●○○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○○○○

Branching Example

```
operation = int(input())
num1 = int(input())
num2 = int(input())
if operation == 1:
    sum_two_numbers(num1, num2)
elif operation == 2:
    multiply_two_numbers(num1, num2)
else:
    divide_two_numbers(num1, num2)
    print('I am here')
```

1. Recap
○○●○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○○○○

Branching Example

```
operation = int(input())
num1 = int(input())
num2 = int(input())
if operation == 1:
    sum_two_numbers(num1, num2)
elif operation == 2:
    multiply_two_numbers(num1, num2)
else:
    divide_two_numbers(num1, num2)
print('I am here')
```

1. Recap
○○○●○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○○○○

Comparison Operators

- <: Strictly less than
- <=: Less than or equal
- >: Strictly greater than
- >=: Greater than or equal
- ==: Equal
- !=: Not equal

```
3.0 == 3  # => True
3.0 >= 3  # => True
# Small-case characters
# have bigger ASCII value
'Aa' > 'aa'  # => False
'hi' == 'hi'  # => True
'a' == None  # => False
3 > 'a'  # => TypeError
3 == 'a'  # => False
```

1. Recap
OOOO●

2. Strings
OOOO

3. While Loops
OO

4. Turtle
OOOOOOO

`bool` **Operators**

How to represent logical operations in Python? (and, or, not)

A	B	A or B	A and B	not A
True	True	True	True	False
True	False	True	False	False
False	True	True	False	True
False	False	False	False	True

`True or False and False` ⇒ **True**

- **and**
- **or**
- **not**

WHY?

1. Recap
○○○○○

2. Strings
●○○○

3. While Loops
○○

4. Turtle
○○○○○○○

Strings

my_string = 'abcde'

<div align="center">

0 1 2 3 4
'a b c d e'
-5 -4 -3 -2 -1

</div>

print(my_string[2]) ⇒ prints c
print(my_string[-2]) ⇒ prints d

Indexing & Slicing

Access specific characters using **indexing**, i.e, **[index]**
Slice strings by using **[start:stop:step]**

```
s = 'Python'
s[1] # => 'y'
s[0:4] # => 'Pyth'
s[:3] # => 'Pyt'
s[3:] # => 'hon'
s[:] # => 'Python'
```

```
s = 'Python'
s[:5:2] # => 'Pto'
s[1:4:3] # => 'y'
s[::3] # => 'Ph'
s[::-1] # => 'nohtyP'
```

1. Recap
ooooo

2. Strings
oo●o

3. While Loops
oo

4. Turtle
ooooooo

String Operations

```
print('This a simple calculator program.')
number1 = input('Please enter the first number:')
number2 = input('Please enter the second number:')
print(f'{number1}+{number2} is {number1 + number2}')
```

```
number1 = int(input('First number:'))
number2 = input('Please enter the second number:')
print(f'{number1}x{number2} is {number1 * number2}')
```

str1 + str2 $\Rightarrow$ **Concatenate** str1 and str2

str1 $\star$ n $\Rightarrow$ Repeat str1 n times.

1. Recap
○○○○○

2. Strings
○○○●

3. While Loops
○○

4. Turtle
○○○○○○○

Example: Evil Laughter

```
https://github.com/koltpython/python-slides/blob/master/Lecture3/
evil_laughter.md
```

While Loops

Repeat some <expression>s as long as a <condition> is True.
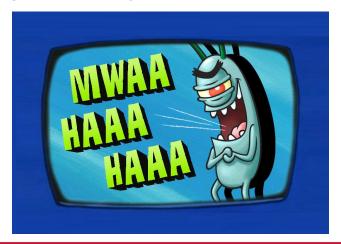
```
while <condition>:
    <expression>
    <expression>
    ...
```

```
x = 15
while x > 10:
    print(x)
    x-=1
```

```
counter = 11
while counter > 6:
    counter -= 1
    print(2**counter)
    counter -= 1
```

<condition> is only checked **before** each execution.

1. Recap
○○○○○

2. Strings
○○○○

3. While Loops
○●

4. Turtle
○○○○○○

Example: Evil Laughter (Cont.)

1. Recap
○○○○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
●○○○○○○

Turtle Module

a Python feature like a drawing board, which lets us command a turtle to draw all over it. . .

1. Recap
○○○○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○●○○○○○

Turtle Functions

```
forward(distance)
```
moves the turtle forward by the specified distance

```
backward(distance)
```
moves the turtle backward by the specified distance

```
pos()
```
returns the turtle's position

1. Recap
OOOOO

2. Strings
OOOO

3. While Loops
OO

4. Turtle
OO●OOOO

Turtle Functions

`setpos(x,y)`
sets the turtle's position to specified x, y coordinates

`right(angle)`
turns the turtle right by angle units

`left(angle)`
turns the turtle left by angle units

1. Recap
OOOOO

2. Strings
OOOO

3. While Loops
OO

4. Turtle
OOO●OOO

Turtle Functions

`setx(x)`
sets the turtle's x coordinate to specified x

`sety(y)`
sets the turtle's y coordinate to specified y

`xcor()`
returns the turtle's x coordinate

1. Recap
○○○○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○●○○

Turtle Functions

`ycor()`
sets the turtle's y coordinate to specified y

`pendown()`
pulls the pen down – drawing when moving.

`penup()`
pulls the pen down – drawing when moving.

Let's draw!

Make the turtle draw 9 squares side by side.
Decompose the task! What about writing a function that draws only one square?

```python
import turtle
def draw_sqr(t,x,y,length):
    t.penup()
    t.setpos(x,y)
    t.pendown()
    k = 0
    while k<4:
        t.forward(length)
        t.right(90)
        k+=1
```

1. Recap
○○○○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○○○●

Let's draw!

How will we make it draw 9 squares by using this function and while loops?

```
my_turtle = turtle.Turtle()
i = 0
x = -200
y = -200
length = 50
while(i<9):
    draw_sqr(my_turtle,x,y,length)
    x+=length
    i+=1
turtle.done()
```