1. Recap
○○○○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○○○

KOLT Python
Branching, While Loops, Turtle Graphics & Strings

Necla Mutlu

Monday 10th February, 2020

KOÇ UNIVERSITY
OFFICE OF LEARNING AND TEACHING

1. Recap
○○○○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○○○○

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)
```

1. Recap
○○●○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○○○○

Branching Example

```python
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)
```

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
○○○○●

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○○○○○

`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` $\Rightarrow$ **True**

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

WHY?

1. Recap
○○○○○

2. Strings
●○○○

3. While Loops
○○

4. Turtle
○○○○○○○

Strings

1. Recap
○○○○○

2. Strings
●○○○

3. While Loops
○○

4. Turtle
○○○○○○○

Strings

```
my_string = 'abcde'
```

1. Recap
○○○○○

2. Strings
●○○○

3. While Loops
○○

4. Turtle
○○○○○○○

Strings

```
my_string = 'abcde'
```

$$0 \; 1 \; 2 \; 3 \; 4$$
$$\text{'a b c d e'}$$

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>

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])
```

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

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])

1. Recap
○○○○○

2. Strings
●○○○

3. While Loops
○○

4. Turtle
○○○○○○○

Strings

`my_string = 'abcde'`

$$\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ \text{'a} & \text{b} & \text{c} & \text{d} & \text{e'} \\ \text{-}5 & \text{-}4 & \text{-}3 & \text{-}2 & \text{-}1 \end{array}$$

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

1. Recap
○○○○○

2. Strings
○●○○

3. While Loops
○○

4. Turtle
○○○○○○○

Indexing & Slicing

1. Recap
OOOOO

2. Strings
O●OO

3. While Loops
OO

4. Turtle
OOOOOOO

Indexing & Slicing

Access specific characters using **indexing**, i.e, `[index]`

1. Recap
○○○○○

2. Strings
○●○○

3. While Loops
○○

4. Turtle
○○○○○○○

Indexing & Slicing

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

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'
```

Indexing & Slicing

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

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

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

1. Recap
○○○○○

2. Strings
○○●○

3. While Loops
○○

4. Turtle
○○○○○○○

String Operations

1. Recap
○○○○○

2. Strings
○○●○

3. While Loops
○○

4. Turtle
○○○○○○○

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}')
```

1. Recap
○○○○○

2. Strings
○○●○

3. While Loops
○○

4. Turtle
○○○○○○○

String Operations

```python
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}')
```

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

1. Recap
OOOOO

2. Strings
OOⱭO

3. While Loops
OO

4. Turtle
OOOOOOO

String Operations

```python
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}')
```

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

str1 + str2 ⇒ **Concatenate** str1 and str2

1. Recap
○○○○○

2. Strings
○○●○

3. While Loops
○○

4. Turtle
○○○○○○○

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 * 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
```

1. Recap
○○○○○

2. Strings
○○○●

3. While Loops
○○

4. Turtle
○○○○○○

Example: Evil Laughter

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

1. Recap
○○○○○

2. Strings
○○○○

3. While Loops
●○

4. Turtle
○○○○○○○

While Loops

While Loops

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

1. Recap
OOOOO

2. Strings
OOOO

3. While Loops
●O

4. Turtle
OOOOOOO

While Loops

Repeat some <expression>s <u>as long as</u> a <condition> is True.

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

1. Recap
OOOOO

2. Strings
OOOO

3. While Loops
●O

4. Turtle
OOOOOOO

While Loops

Repeat some <expression>s <u>as long as</u> a <condition> is True.

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

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

1. Recap
○○○○○

2. Strings
○○○○

3. While Loops
●○

4. Turtle
○○○○○○○

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
```

1. Recap
OOOOO

2. Strings
OOOO

3. While Loops
●O

4. Turtle
OOOOOOO

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

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

Turtle Functions

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

1. Recap
OOOOO

2. Strings
OOOO

3. While Loops
OO

4. Turtle
O●OOOOO

Turtle Functions

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

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

1. Recap
OOOOO

2. Strings
OOOO

3. While Loops
OO

4. Turtle
O●OOOOO

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

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

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
○○○○○

2. Strings
○○○○

3. While Loops
○○

4. Turtle
○○○●○○○

Turtle Functions

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

Turtle Functions

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

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

1. Recap
ooooo

2. Strings
oooo

3. While Loops
oo

4. Turtle
oooooooo

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
OOOOO

2. Strings
OOOO

3. While Loops
OO

4. Turtle
OOOO●OO

Turtle Functions

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

1. Recap
OOOOO

2. Strings
OOOO

3. While Loops
OO

4. Turtle
OOOO●OO

Turtle Functions

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

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

1. Recap
00000

2. Strings
0000

3. While Loops
00

4. Turtle
0000●00

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.

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?

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
```

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
```

Let's draw!

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

1. Recap
OOOOO

2. Strings
OOOO

3. While Loops
OO

4. Turtle
OOOOOO●

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()
```

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()
```