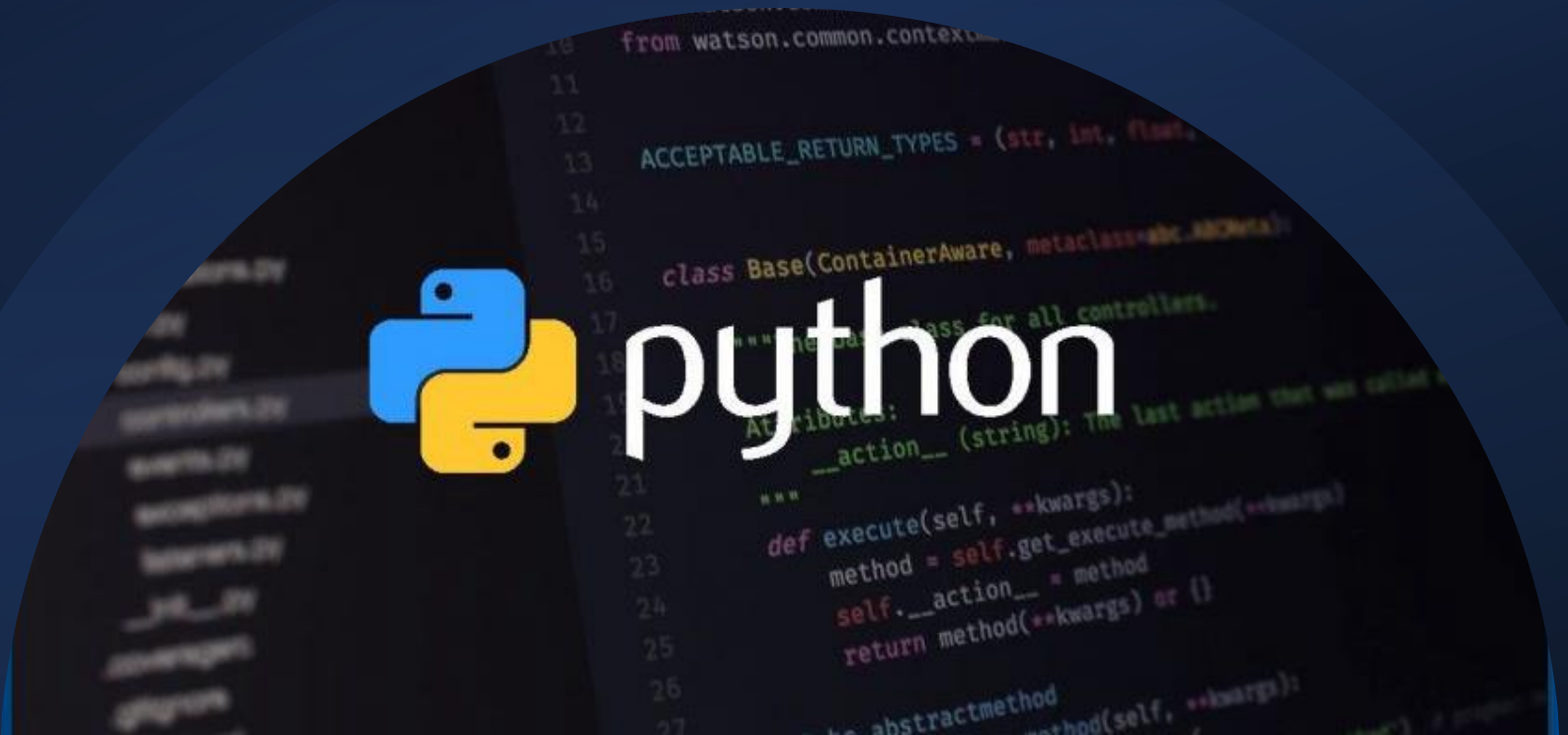
Learning python

202201019



4110E234

SYLLABUS

- Python Conditions and If statements
- Python For Loops
- Python While Loops

• Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- These conditions can be used in several ways, most commonly in "if statements" and loops.

These conditions can be used in several ways, most commonly in "if statements" and loops.

- An "if statement" is written by using the if keyword.

Python Conditions and If statements

If statement:

```
a = 89
b = 169

if b > a:
  print("b is greater than a")
```

```
b is greater than a
```

- In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 89, and b is 169, we know that 169 is greater than 89, and so we print to screen that "b is greater than a".

• Indentation

- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Indentation

If statement, without indentation (will raise an error):

```
[2]  a = 89
     b = 169
     if b > a:
     print("b is greater than a") # you will get an error
```

```
  File "<ipython-input-2-ed2bc87ad356>", line 4
    print("b is greater than a") # you will get an error
        ^
IndentationError: expected an indented block
```

SEARCH STACK OVERFLOW

• Elif

- The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Elif

```
a = 69
b = 69
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

```
a and b are equal
```

- In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

• Else

- The else keyword catches anything which isn't caught by the preceding conditions.

Else

```python
a = 169
b = 89
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```

- In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

• Else

- You can also have an else without the elif:

```
a = 169
b = 89
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

```
b is not greater than a
```

• Short Hand If

- If you have only one statement to execute, you can put it on the same line as the if statement.

Short Hand If

One line if statement:

```
a = 169
b = 89

if a > b: print("a is greater than b")
```

```
a is greater than b
```

• Short Hand If … Else

- If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Short Hand If ... Else

One line if else statement:

```
a = 9
b = 371

print("A") if a > b else print("B")
```

```
B
```

- This technique is known as **Ternary Operators**, or **Conditional Expressions**.

• Short Hand If … Else

- You can also have multiple else statements on the same line

One line if else statement, with 3 conditions:

```
[8]  a = 371
     b = 371

     print("A") if a > b else print("=") if a == b else print("B")
```

```
=
```

• And

- The and keyword is a logical operator, and is used to combine conditional statements:

And

Test if a is greater than b, AND if c is greater than a:

```
a = 390
b = 69
c = 750
if a > b and c > a:
  print("Both conditions are True")
```

```
Both conditions are True
```

• Or

- The or keyword is a logical operator, and is used to combine conditional statements:

Or

Test if a is greater than b, OR if a is greater than c:

```
a = 390
b = 69
c = 750
if a > b or a > c:
  print("At least one of the conditions is True")
```

```
At least one of the conditions is True
```

• Nested If

- You can have if statements inside if statements, this is called *nested* if statements.

Nested If

```python
x = 61

if x > 30:
  print("Above thirty,")
  if x > 50:
    print("and also above 50!")
  else:
    print("but not above 50.")
```

```
Above thirty,
and also above 50!
```

• The pass Statement

- if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

The pass Statement

```python
a = 89
b = 169

if b > a:
  pass

# having an empty if statement like this, would raise an error without the pass statement
```

• Python For Loops

Python For Loops

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

- **Print each classmate in a classmate list:**

Python For Loops

Print each fruit in a fruit list:

```
classmates = ["james", "kenneth", "raive"]
for x in classmates:
  print(x)
```

```
james
kenneth
raive
```

The for loop does not require an indexing variable to set beforehand.

• Looping Through a String

- Even strings are iterable objects, they contain a sequence of characters:

Looping Through a String

Loop through the letters in the word "angelle":

```
for x in "angelle":
  print(x)
```

```
a
n
g
e
l
l
e
```

• The break Statement

- With the break statement we can stop the loop before it has looped through all the items:

The break Statement

Exit the loop when x is "kenneth":

```
classmates = ["james", "kenneth", "raive"]
for x in classmates:
  print(x)
  if x == "kenneth":
    break
```

```
james
kenneth
```

• The break Statement

Exit the loop when x is "kenneth", but this time the break comes before the print:

```
classmates = ["james", "kenneth", "raive"]
for x in classmates:
  if x == "kenneth":
    break
  print(x)
```

```
james
```

• The continue Statement

- With the continue statement we can stop the current iteration of the loop, and continue with the next:

The continue Statement

Do not print kenneth:

```
classmates = ["james", "kenneth", "raive"]
for x in classmates:
  if x == "kenneth":
    continue
  print(x)
```

```
james
raive
```

• The range() Function

- To loop through a set of code a specified number of times, we can use the range() function,
- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

The range() Function

Using the range() function:

```
for x in range(9):
  print(x)
```

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

- Note that range(9) is not the values of 0 to 9, but the values 0 to 6.

• The range() Function

- The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(3, 9), which means values from 3 to 9 (but not including 9):

Using the start parameter:

```
for x in range(3, 9):
  print(x)
```

```
3
4
5
6
7
8
```

• The range() Function

- The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(6, 90, 3):

Increment the sequence with 3 (default is 1):

```python
for x in range(6, 90, 3):
  print(x)
```

```
6
9
12
15
18
21
24
27
30
33
36
39
42
45
48
51
54
57
60
63
66
69
72
75
78
81
84
87
```

• Else in for Loop

- The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

Else in For Loop

Print all numbers from 0 to 8, and print a message when the loop has ended:

```python
for x in range(9):
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
4
5
6
7
8
Finally finished!
```

Note: The `else` block will NOT be executed if the loop is stopped by a `break` statement.

• Else in for Loop

Break the loop when x is 6, and see what happens with the else block:

```
for x in range(9):
  if x == 6: break
  print(x)
else:
  print("Finally finished!")

#If the loop breaks, the else block is not executed.
```

```
0
1
2
3
4
5
```

• Nested Loops

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":

Nested Loops

Print each adjective for every classmates:

```python
adj = ["smart", "tall", "funny"]
classmates = ["james", "kenneth", "raive"]

for x in adj:
  for y in classmates:
    print(x, y)
```

```
smart james
smart kenneth
smart raive
tall james
tall kenneth
tall raive
funny james
funny kenneth
funny raive
```

• The pass Statement

- for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

The pass Statement

```
for x in [0, 1, 2]:
  pass

# having an empty for loop like this, would raise an error without the pass statement
```

• The while Loop

- With the while loop we can execute a set of statements as long as a condition is true.

Print i as long as i is less than 9:

```
i = 1
while i < 9:
  print(i)
  i += 1
```

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

Note: remember to increment i, or else the loop will continue forever.

- The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

• The break Statement

- With the break statement we can stop the loop even if the while condition is true:

The break Statement

Exit the loop when i is 6:

```
i = 1
while i < 9:
  print(i)
  if (i == 6):
    break
  i += 1
```

```
1
2
3
4
5
6
```

• The continue Statement

- With the continue statement we can stop the current iteration, and continue with the next:

The continue Statement

Exit the loop when i is 6:

```
i = 1
while i < 9:
  print(i)
  if (i == 6):
    break
  i += 1
```

```
1
2
3
4
5
6
```

• The else Statement

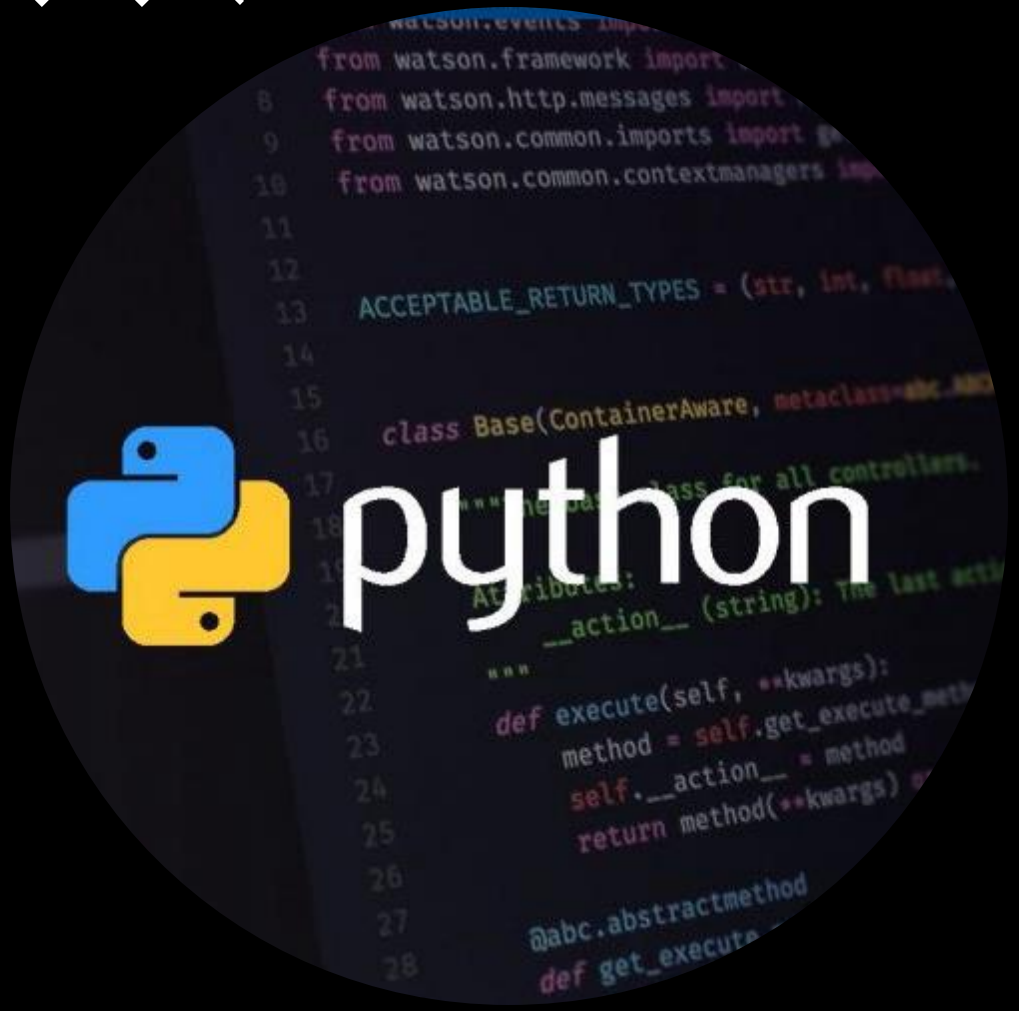
- With the else statement we can run a block of code once when the condition no longer is true:

The else Statement

Print a message once the condition is false:

```
i = 1
while i < 9:
  print(i)
  i += 1
else:
  print("i is no longer less than 9")
```

```
1
2
3
4
5
6
7
8
i is no longer less than 9
```



THANK YOU!

CHINESE NAME: 喬萬斯

STUDENT ID: 4110E234

English Name: Joaquin Vasti R. Rapada

nickname: Wax

email: vastiplayer@gmail.com

GITHUB: https://github.com/4110E34/cs20220921/blob/main/python/1019.md