Programming for Management Studies

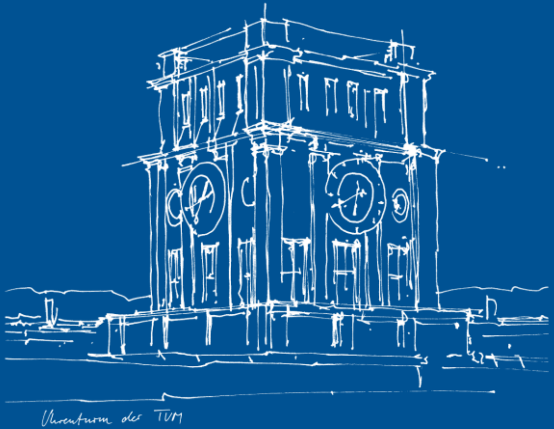
Part I - Lecture 02

Control flow, Statements, Exceptions handling

Lecturer: Maotong Sun

Technische Universität München

TUM Campus Heilbronn



Reference book:

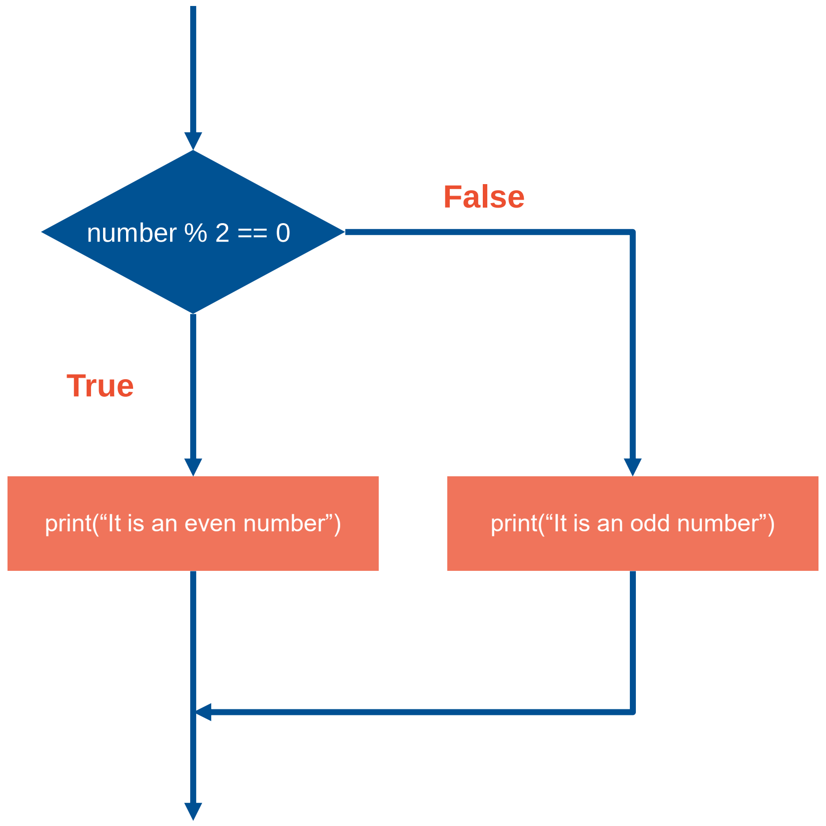
- *Python Distilled (David Beazley)*
- *Python Crash Course (Eric Matthes)*

Control flow

Examples

```python
number = 95
if number % 2 == 0:
    print("It is an even number")
else:
    print("It is an odd number")
```

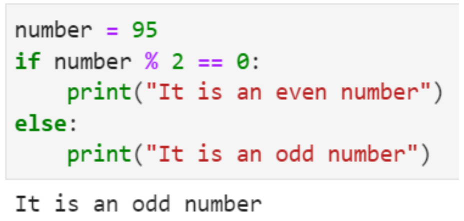
```
It is an odd number
```



Control flow

Basics

- Python programs are structured as a sequence of statements.
- Different levels of statements are defined using colons and indentations (whitespace).



```python
number = 95
if number % 2 == 0:
    print("It is an even number")
else:
    print("It is an odd number")
```

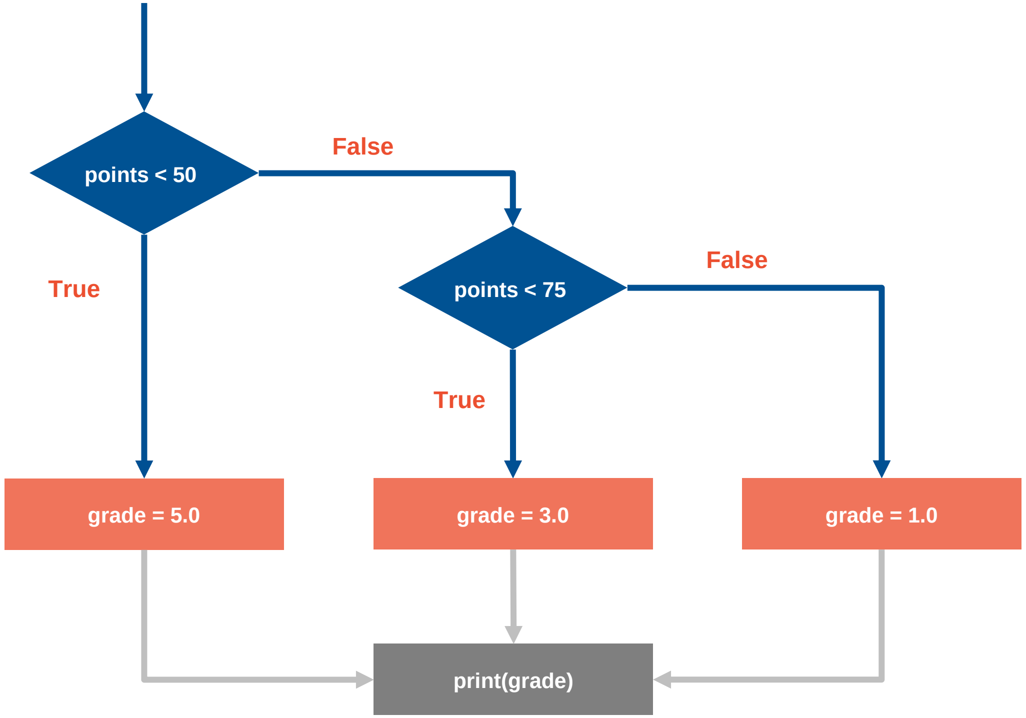
```
It is an odd number
```

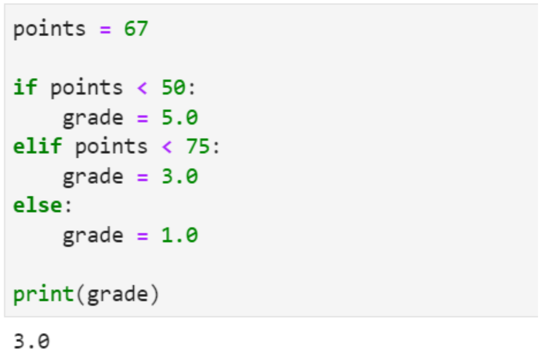
- We often want to execute certain blocks of codes when some conditions are met.
  - If true then do this, otherwise do ...
  - While this is still true, do …

Statements

If, elif, else

- The if, else, and elif statements control conditional code execution.





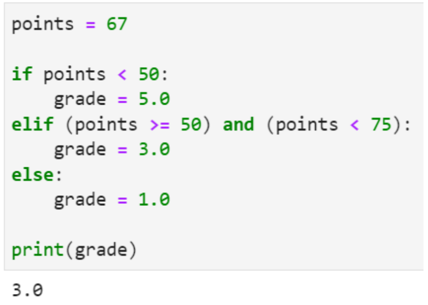
```
points = 67

if points < 50:
    grade = 5.0
elif points < 75:
    grade = 3.0
else:
    grade = 1.0

print(grade)
```

```
3.0
```

Statements

If, elif, else

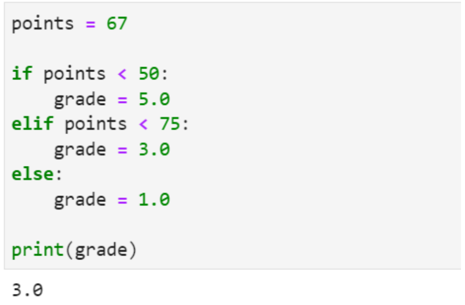
- The rest of the if…elif...else chain is skipped when a condition is met.



```
points = 67

if points < 50:
    grade = 5.0
elif (points >= 50) and (points < 75):
    grade = 3.0
else:
    grade = 1.0

print(grade)
```

```
3.0
```

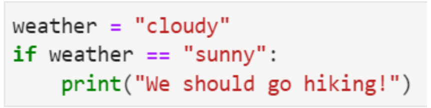
=



```
points = 67

if points < 50:
    grade = 5.0
elif points < 75:
    grade = 3.0
else:
    grade = 1.0

print(grade)
```

```
3.0
```

- The elif and else statements are optional (if no operations have to be performed when the condition is not met).

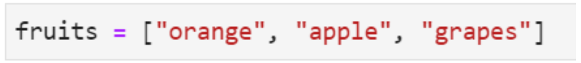


```
weather = "cloudy"
if weather == "sunny":
    print("We should go hiking!")
```

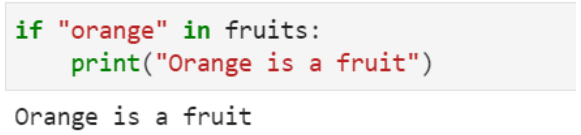
Statements

If, elif, else – other usages

- Checking if a value is in a list (or tuple, set, dictionary keys):

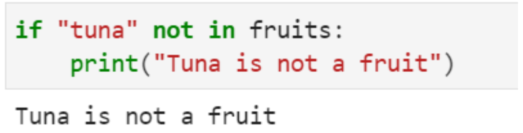


```python
fruits = ["orange", "apple", "grapes"]
```



```python
if "orange" in fruits:
    print("Orange is a fruit")
```

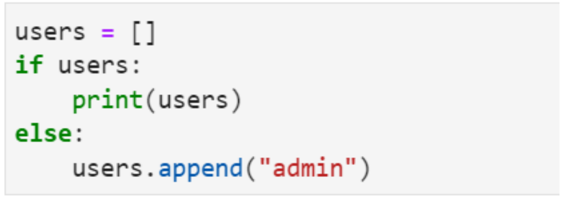
```
Orange is a fruit
```



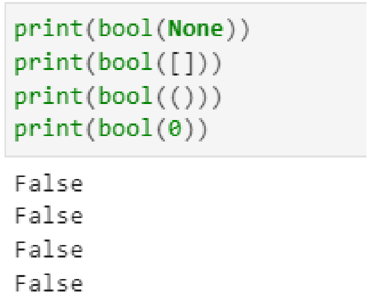
```python
if "tuna" not in fruits:
    print("Tuna is not a fruit")
```

```
Tuna is not a fruit
```

- Checking if a list (or tuple, set) is empty:



```python
users = []
if users:
    print(users)
else:
    users.append("admin")
```



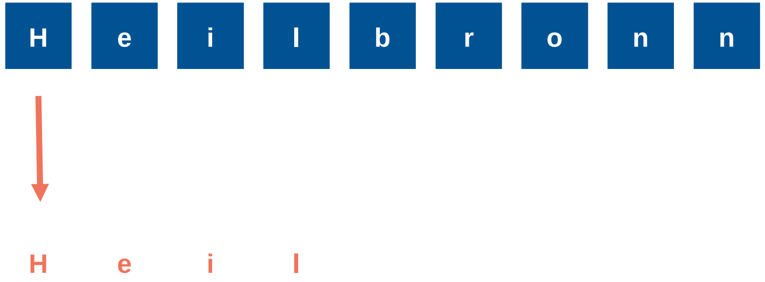
```python
print(bool(None))
print(bool([]))
print(bool(()))
print(bool(0))
```

```
False
False
False
False
```

Statements

For loops

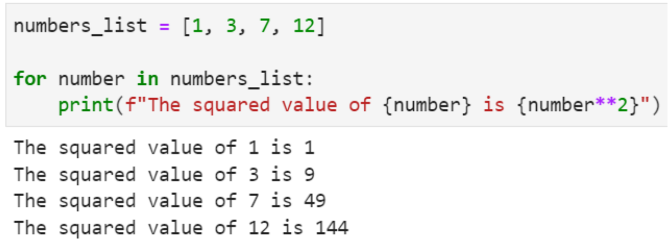
- In Python, an object is called iterable if we can iterate over the object and return members one at a time.
    - Example: strings, lists, dictionaries, tuples



Statements

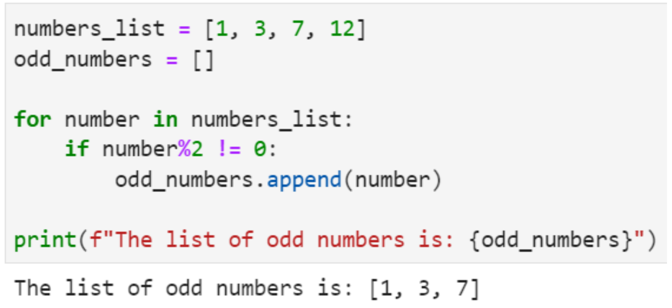
For loops

- For loops are often used to iterate through iterable objects and perform some operations with the returned values.
- The basic syntax of a for loop:
  - Note the indentations!



```
numbers_list = [1, 3, 7, 12]

for number in numbers_list:
    print(f"The squared value of {number} is {number**2}")
```

```
The squared value of 1 is 1
The squared value of 3 is 9
The squared value of 7 is 49
The squared value of 12 is 144
```



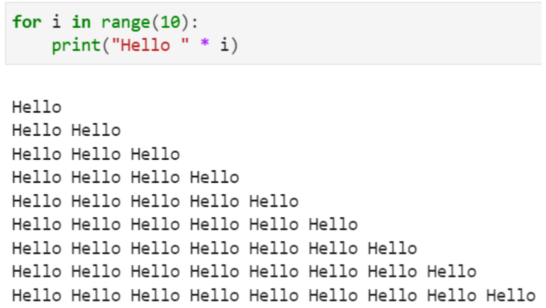
```
numbers_list = [1, 3, 7, 12]
odd_numbers = []

for number in numbers_list:
    if number%2 != 0:
        odd_numbers.append(number)

print(f"The list of odd numbers is: {odd_numbers}")
```

```
The list of odd numbers is: [1, 3, 7]
```

Statements

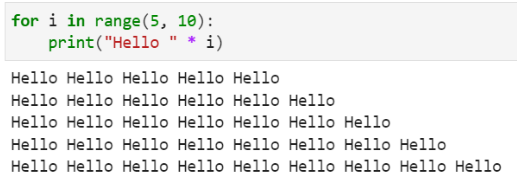
For loops

- When we want to perform an operation a number of times, use the range() function:



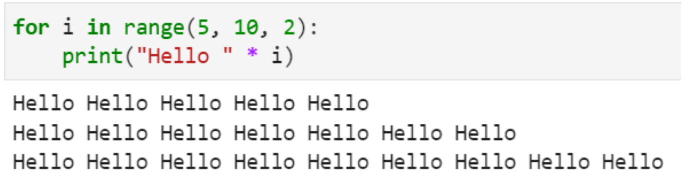
```python
for i in range(10):
    print("Hello " * i)
```

```

Hello
Hello Hello
Hello Hello Hello
Hello Hello Hello Hello
Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello Hello Hello Hello
```



```python
for i in range(5, 10):
    print("Hello " * i)
```

```
Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello Hello Hello Hello
```



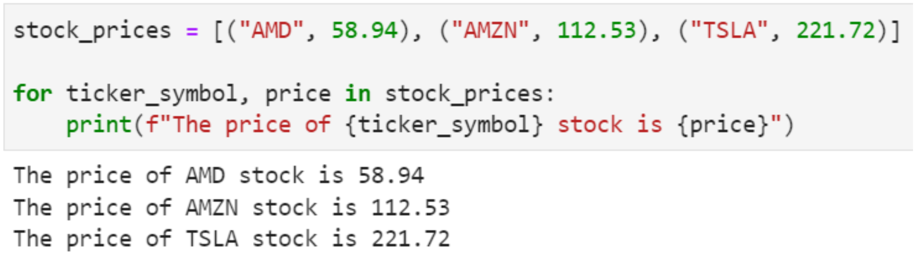
```python
for i in range(5, 10, 2):
    print("Hello " * i)
```

```
Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello Hello Hello Hello
```

Statements

For loops

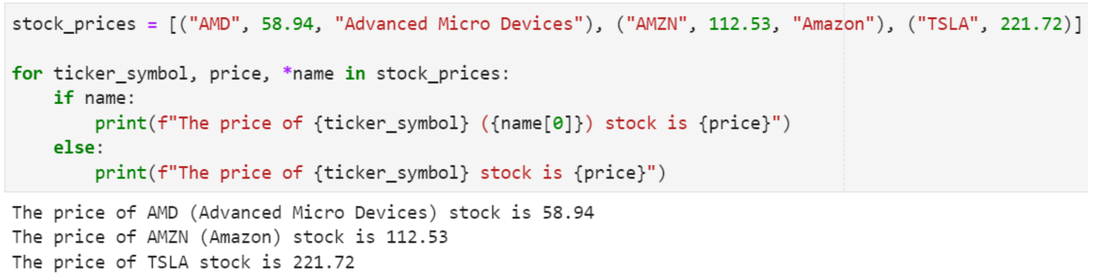
- Tuples unpacking operations can be used when iterating over a list of tuples.



```
stock_prices = [("AMD", 58.94), ("AMZN", 112.53), ("TSLA", 221.72)]

for ticker_symbol, price in stock_prices:
    print(f"The price of {ticker_symbol} stock is {price}")
```

```
The price of AMD stock is 58.94
The price of AMZN stock is 112.53
The price of TSLA stock is 221.72
```

- Wildcard unpacking * is used when the number of items to be unpacked is not known.



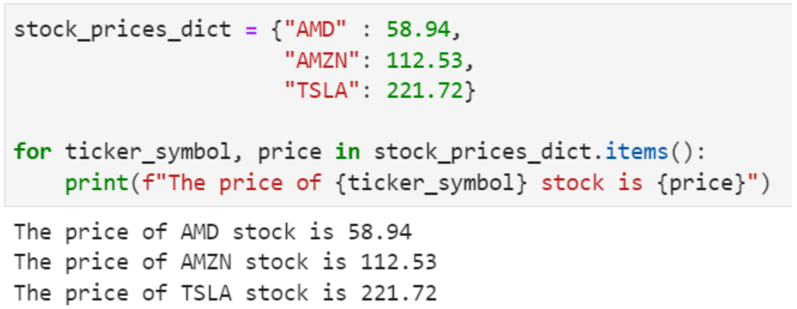
```
stock_prices = [("AMD", 58.94, "Advanced Micro Devices"), ("AMZN", 112.53, "Amazon"), ("TSLA", 221.72)]

for ticker_symbol, price, *name in stock_prices:
    if name:
        print(f"The price of {ticker_symbol} ({name[0]}) stock is {price}")
    else:
        print(f"The price of {ticker_symbol} stock is {price}")
```

```
The price of AMD (Advanced Micro Devices) stock is 58.94
The price of AMZN (Amazon) stock is 112.53
The price of TSLA stock is 221.72
```

Statements

For loops

- A dictionary can be iterated using the following syntax:



```python
stock_prices_dict = {"AMD" : 58.94,
                     "AMZN": 112.53,
                     "TSLA": 221.72}

for ticker_symbol, price in stock_prices_dict.items():
    print(f"The price of {ticker_symbol} stock is {price}")
```

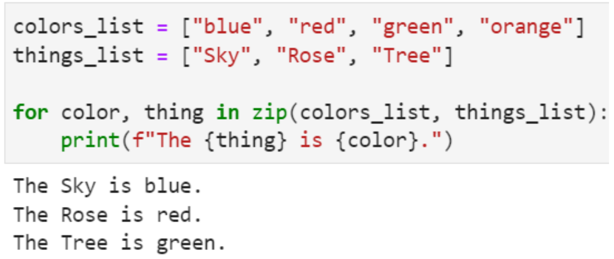
```
The price of AMD stock is 58.94
The price of AMZN stock is 112.53
The price of TSLA stock is 221.72
```

- Keys and values can be iterated separately by replacing items() with keys() or values().
- Note: when iterating a dictionary at different times, the same ordering is not guaranteed and, therefore, should not be relied upon!

Statements

For loops – advanced usages

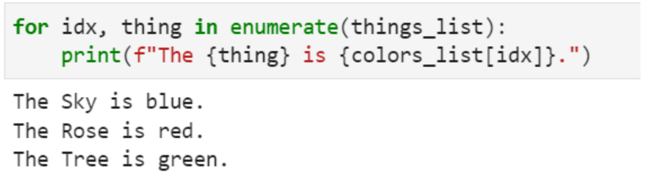
- Two (or more) iterables can be iterated at the same time using the zip() function.



```
colors_list = ["blue", "red", "green", "orange"]
things_list = ["Sky", "Rose", "Tree"]

for color, thing in zip(colors_list, things_list):
    print(f"The {thing} is {color}.")
```

```
The Sky is blue.
The Rose is red.
The Tree is green.
```

- The current iteration number can be extracted using the enumerate() function:



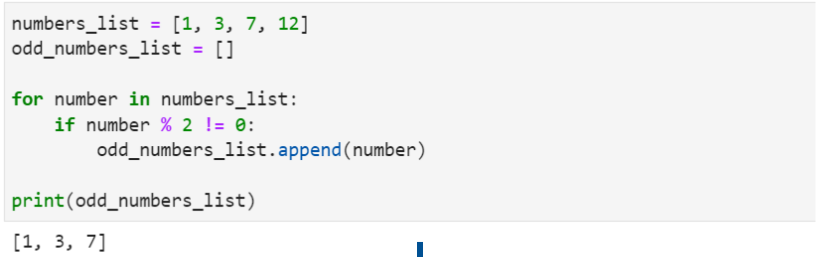
```
for idx, thing in enumerate(things_list):
    print(f"The {thing} is {colors_list[idx]}.")
```

```
The Sky is blue.
The Rose is red.
The Tree is green.
```

Statements

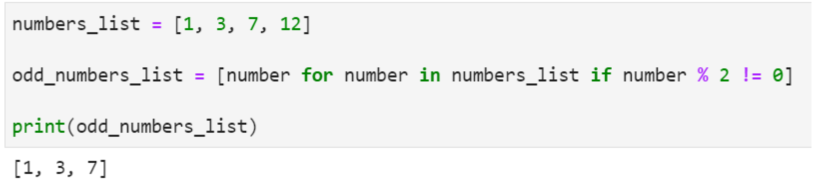
For loops – List comprehensions

- List comprehension is one of the “syntactic sugar” in Python.
  - allow short, clean, and readable code.



```
numbers_list = [1, 3, 7, 12]
odd_numbers_list = []

for number in numbers_list:
    if number % 2 != 0:
        odd_numbers_list.append(number)

print(odd_numbers_list)
```

```
[1, 3, 7]
```



```
numbers_list = [1, 3, 7, 12]

odd_numbers_list = [number for number in numbers_list if number % 2 != 0]

print(odd_numbers_list)
```

```
[1, 3, 7]
```

Statements

For loops – List comprehensions

- Example:
  - Extract the list of words from a sentence that starts with the letter “t”.

```
sentence = "Digitalization is transforming vast areas of our live and new \
            technologies are fundamentally changing the way companies work."
```

```
words_list = []

for word in sentence.split(" "):
    if word.startswith("t"):
        words_list.append(word)

print(words_list)
```

```
['transforming', 'technologies', 'the']
```

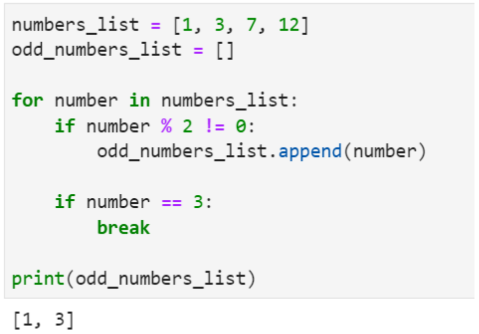
```
[word for word in sentence.split(" ") if word.startswith("t")]
```

```
['transforming', 'technologies', 'the']
```

Statements

For loops – advanced usages

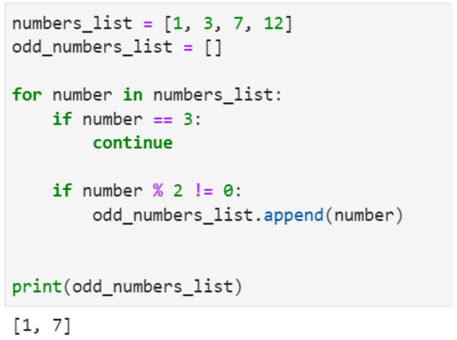
- A loop can be terminated prematurely using the command break.
    - Note that the break command only terminates the closest enclosing loop.



```python
numbers_list = [1, 3, 7, 12]
odd_numbers_list = []

for number in numbers_list:
    if number % 2 != 0:
        odd_numbers_list.append(number)

    if number == 3:
        break

print(odd_numbers_list)
```

```
[1, 3]
```

- continue: go to the top of the closest enclosing loop.
- pass: do nothing.



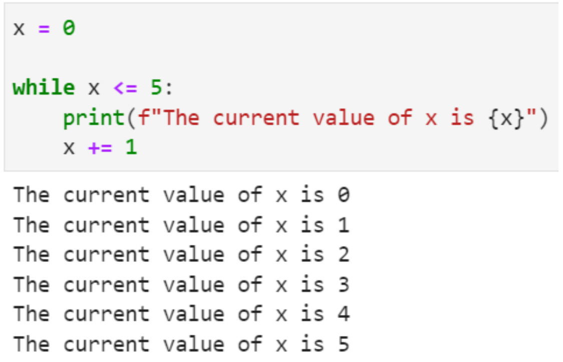
```python
numbers_list = [1, 3, 7, 12]
odd_numbers_list = []

for number in numbers_list:
    if number == 3:
        continue

    if number % 2 != 0:
        odd_numbers_list.append(number)


print(odd_numbers_list)
```

```
[1, 7]
```

Statements

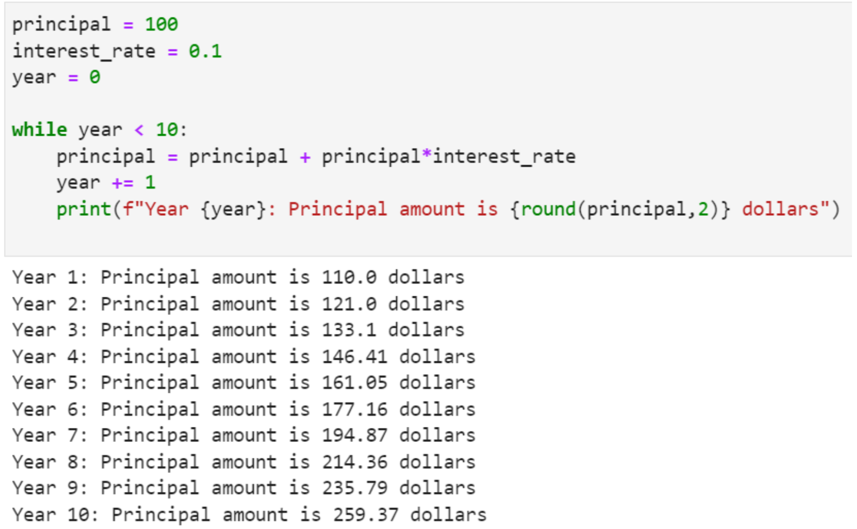
While loops

- While loops are used when we want to execute a block of code as long as some conditions remain True.



```
x = 0

while x <= 5:
    print(f"The current value of x is {x}")
    x += 1
```

```
The current value of x is 0
The current value of x is 1
The current value of x is 2
The current value of x is 3
The current value of x is 4
The current value of x is 5
```



```
principal = 100
interest_rate = 0.1
year = 0

while year < 10:
    principal = principal + principal*interest_rate
    year += 1
    print(f"Year {year}: Principal amount is {round(principal,2)} dollars")
```

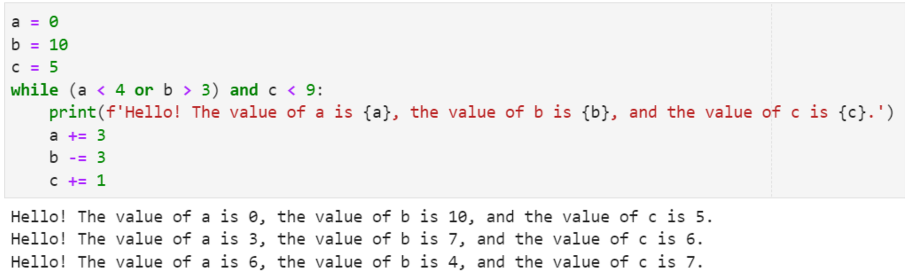
```
Year 1: Principal amount is 110.0 dollars
Year 2: Principal amount is 121.0 dollars
Year 3: Principal amount is 133.1 dollars
Year 4: Principal amount is 146.41 dollars
Year 5: Principal amount is 161.05 dollars
Year 6: Principal amount is 177.16 dollars
Year 7: Principal amount is 194.87 dollars
Year 8: Principal amount is 214.36 dollars
Year 9: Principal amount is 235.79 dollars
Year 10: Principal amount is 259.37 dollars
```

Statements

While loops

- Loop control commands such as continue, break, and pass can also be used in the same way as for loops.

- Multiple conditions can be combined:



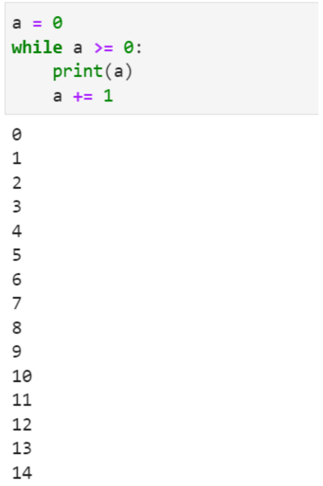
```
a = 0
b = 10
c = 5
while (a < 4 or b > 3) and c < 9:
    print(f'Hello! The value of a is {a}, the value of b is {b}, and the value of c is {c}.')
    a += 3
    b -= 3
    c += 1
```

```
Hello! The value of a is 0, the value of b is 10, and the value of c is 5.
Hello! The value of a is 3, the value of b is 7, and the value of c is 6.
Hello! The value of a is 6, the value of b is 4, and the value of c is 7.
```

Statements

While loops

- Make sure that the stopping condition is met at some point in time. Otherwise the while loop will be infinite.



```
a = 0
while a >= 0:
    print(a)
    a += 1
```

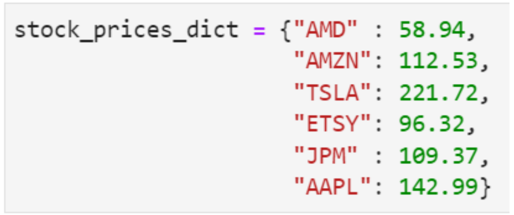
```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

- Infinite loop can be stopped using the Ctrl + C command from the keyboard.

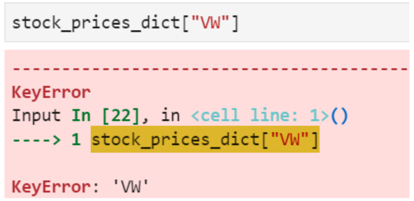
Exceptions handling

try - except

- Exceptions indicate errors and break out of the normal control flow of a program.
- To catch an exception, use the try and except statements.

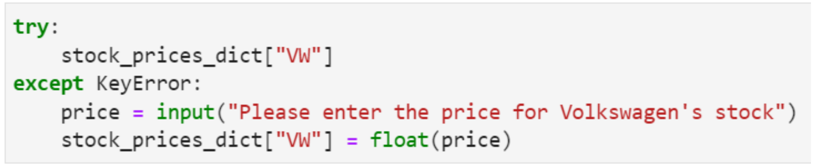


```python
stock_prices_dict = {"AMD" : 58.94,
                     "AMZN": 112.53,
                     "TSLA": 221.72,
                     "ETSY": 96.32,
                     "JPM" : 109.37,
                     "AAPL": 142.99}
```



```python
stock_prices_dict["VW"]
```

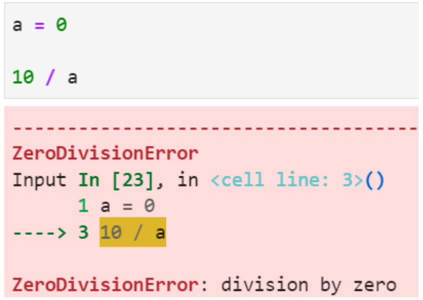
```
---------------------------------------
KeyError
Input In [22], in <cell line: 1>()
----> 1 stock_prices_dict["VW"]

KeyError: 'VW'
```



```python
try:
    stock_prices_dict["VW"]
except KeyError:
    price = input("Please enter the price for Volkswagen's stock")
    stock_prices_dict["VW"] = float(price)
```

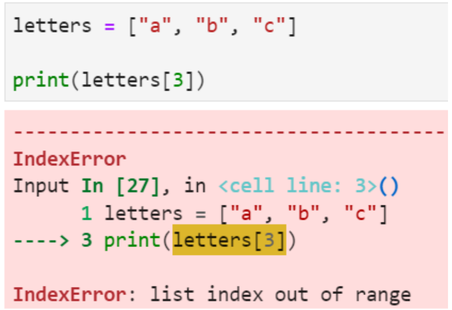
Exceptions handling

Examples



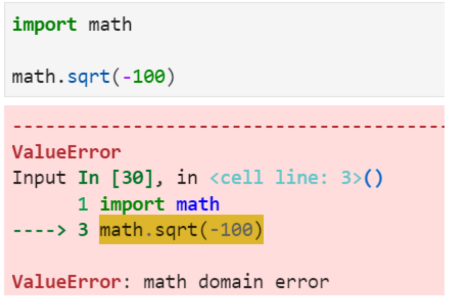
```
a = 0

10 / a
```

```
-------------------------------------
ZeroDivisionError
Input In [23], in <cell line: 3>()
      1 a = 0
----> 3 10 / a

ZeroDivisionError: division by zero
```



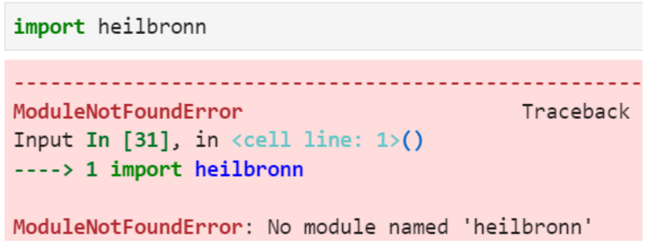
```
letters = ["a", "b", "c"]

print(letters[3])
```

```
--------------------------------------
IndexError
Input In [27], in <cell line: 3>()
      1 letters = ["a", "b", "c"]
----> 3 print(letters[3])

IndexError: list index out of range
```



```
import math

math.sqrt(-100)
```

```
---------------------------------------
ValueError
Input In [30], in <cell line: 3>()
      1 import math
----> 3 math.sqrt(-100)

ValueError: math domain error
```



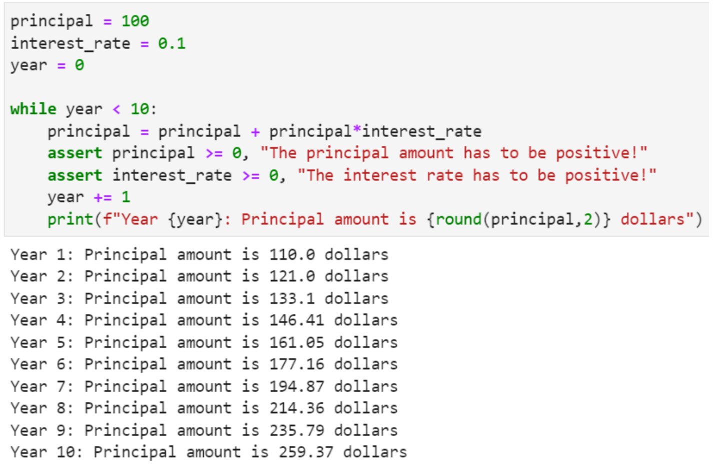
```
import heilbronn
```

```
---------------------------------------------------------
ModuleNotFoundError                       Traceback
Input In [31], in <cell line: 1>()
----> 1 import heilbronn

ModuleNotFoundError: No module named 'heilbronn'
```

Exceptions handling

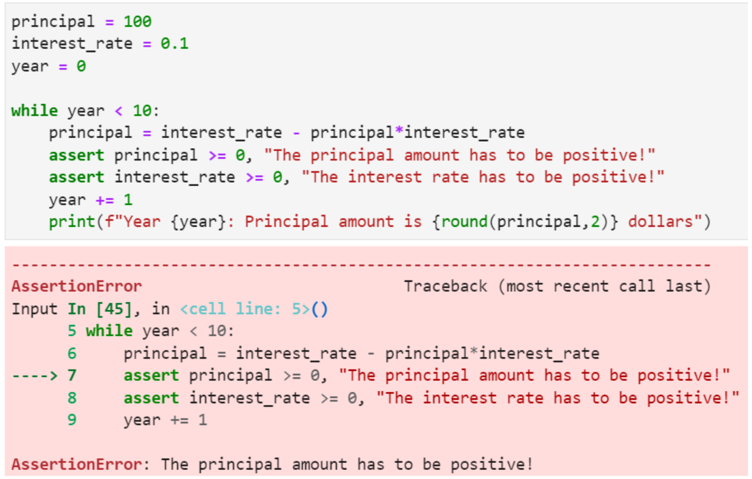
Assertions

- The assert statement help in debugging your code and avoiding obvious mistakes passing silently.
- Use assert as a logic safeguard for your code.



```
principal = 100
interest_rate = 0.1
year = 0

while year < 10:
    principal = principal + principal*interest_rate
    assert principal >= 0, "The principal amount has to be positive!"
    assert interest_rate >= 0, "The interest rate has to be positive!"
    year += 1
    print(f"Year {year}: Principal amount is {round(principal,2)} dollars")
```

```
Year 1: Principal amount is 110.0 dollars
Year 2: Principal amount is 121.0 dollars
Year 3: Principal amount is 133.1 dollars
Year 4: Principal amount is 146.41 dollars
Year 5: Principal amount is 161.05 dollars
Year 6: Principal amount is 177.16 dollars
Year 7: Principal amount is 194.87 dollars
Year 8: Principal amount is 214.36 dollars
Year 9: Principal amount is 235.79 dollars
Year 10: Principal amount is 259.37 dollars
```



```
principal = 100
interest_rate = 0.1
year = 0

while year < 10:
    principal = interest_rate - principal*interest_rate
    assert principal >= 0, "The principal amount has to be positive!"
    assert interest_rate >= 0, "The interest rate has to be positive!"
    year += 1
    print(f"Year {year}: Principal amount is {round(principal,2)} dollars")
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [45], in <cell line: 5>()
      5 while year < 10:
      6     principal = interest_rate - principal*interest_rate
----> 7     assert principal >= 0, "The principal amount has to be positive!"
      8     assert interest_rate >= 0, "The interest rate has to be positive!"
      9     year += 1

AssertionError: The principal amount has to be positive!
```

Question?