



Προγραμματισμός και Εφαρμογές με την Python Turtle

Δαδαλιάρης Αντώνιος (dadaliaris@uth.gr)

Περιεχόμενα

- Python Basics
 - History
 - Characteristics
 - Applications
 - Variables & Numbers
 - Loops
 - Lists, Strings, Tuples, Sets & Dictionaries
 - Functions
 - Object Oriented Programming
 - File Handling
 - Python Basics Self-Evaluation Test
- Python Ecosystem
 - Matplotlib
 - Turtle

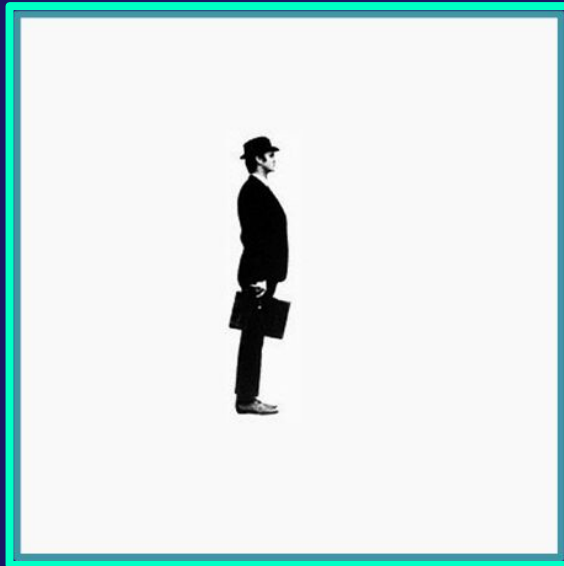


01

Python Basics Recap

“Pythonic” History

- Created by Guido Van Rossum in 1989 as a Christmas holiday project. . . .
- Current Version: 3.10.1
- Named after Monty Python



Python Characteristics

- Dynamic
- Open-source
- High-level
- Interpreted
- Object-oriented
- General-purpose
- OS agnostic
- Multi-Paradigm
- Frameworks
- Extendable

Python Characteristics

- Dynamic
 - Data can be modified dynamically during execution.
- Open-source
- High-level
- Interpreted
- Object-oriented
- General-purpose
- OS agnostic
- Multi-Paradigm
- Frameworks
- Extendable

Python Characteristics

- Dynamic
- Open-source
 - OSI licensed
- High-level
- Interpreted
- Object-oriented
- General-purpose
- OS agnostic
- Multi-Paradigm
- Frameworks
- Extendable

Python Characteristics

- Dynamic
- Open-source
- High-level
 - Syntactic and grammar rules that enhance readability & understanding.
- Interpreted
- Object-oriented
- General-purpose
- OS agnostic
- Multi-Paradigm
- Frameworks
- Extendable

Python Characteristics

- Dynamic
- Open-source
- High-level
- Interpreted
 - “Serial” command execution.
- Object-oriented
- General-purpose
- OS agnostic
- Multi-Paradigm
- Frameworks
- Extendable

Python Characteristics

- Dynamic
- Open-source
- High-level
- Interpreted
- Object-oriented
 - Classes
 - Objects
 - “Loose” encapsulation rules
- General-purpose
- OS agnostic
- Multi-Paradigm
- Frameworks
- Extendable

Python Characteristics

- Dynamic
- Open-source
- High-level
- Interpreted
- Object-oriented
- General-purpose
 - Various applications
- OS agnostic
- Multi-Paradigm
- Frameworks
- Extendable

Python Characteristics

- Dynamic
- Open-source
- High-level
- Interpreted
- Object-oriented
- General-purpose
- OS agnostic
 - Available for almost all operating systems.
- Multi-Paradigm
- Frameworks
- Extendable

Python Characteristics

- Dynamic
- Open-source
- High-level
- Interpreted
- Object-oriented
- General-purpose
- OS agnostic
- Multi-Paradigm
 - Procedural programming
 - Imperative programming
 - Object-oriented programming
 - Functional programming
- Frameworks
- Extendable

Python Characteristics

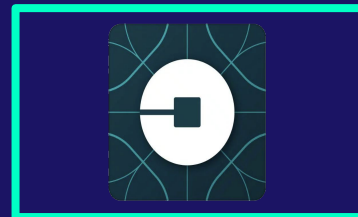
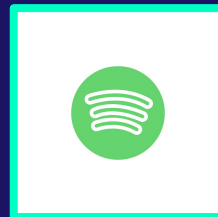
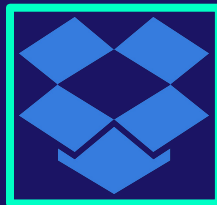
- Dynamic
- Open-source
- High-level
- Interpreted
- Object-oriented
- General-purpose
- OS agnostic
- Multi-Paradigm
- Frameworks
 - Flask
 - Django
 -
- Extendable

Python Characteristics

- Dynamic
- Open-source
- High-level
- Interpreted
- Object-oriented
- General-purpose
- OS agnostic
- Multi-Paradigm
- Frameworks
- Extendable
 - Cython
 - Jython
 - Iron Python
 -

Applications built with Python

- Google
- Instagram
- Dropbox
- Pinterest
- Spotify
- Uber
- Reddit
- Netflix



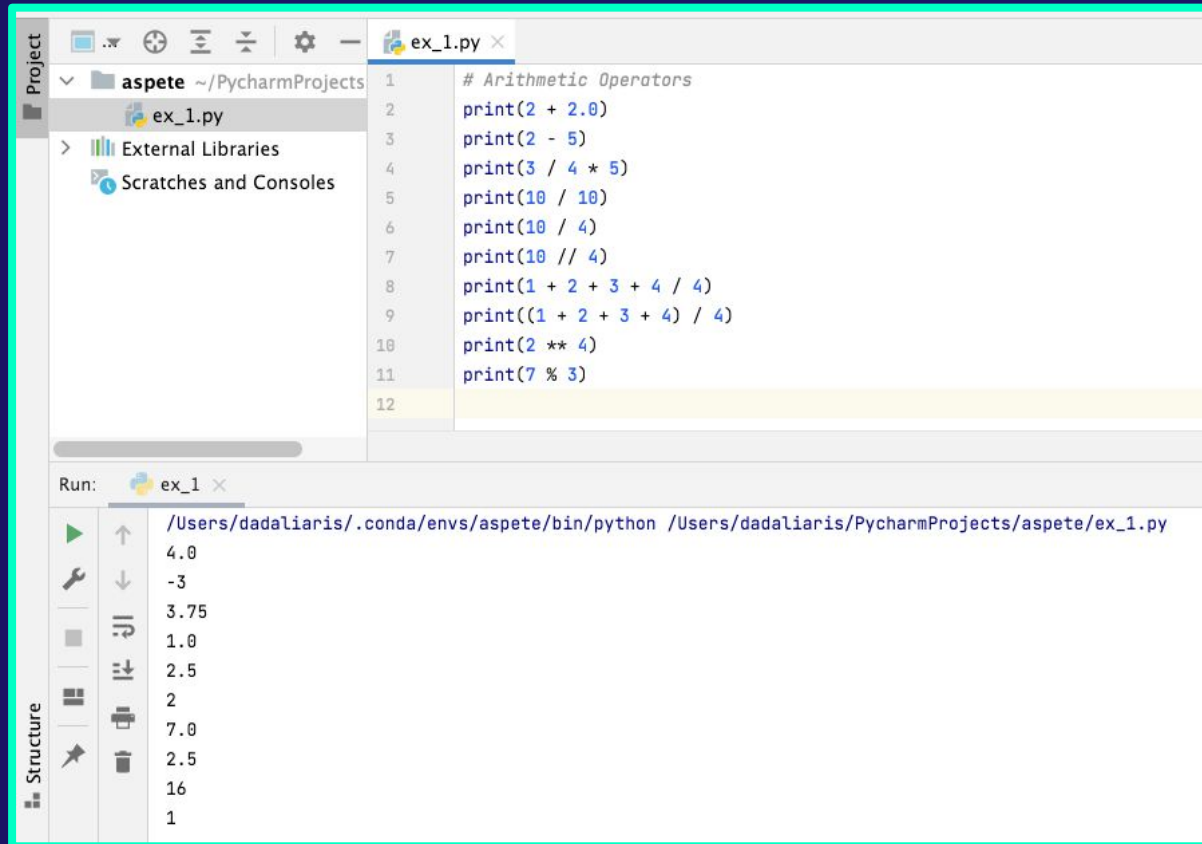
Variables & Numbers (1)

- Python Console (IDLE, Spyder,):
 - We can perform any arithmetic calculation directly at the console.
 - We can also execute various commands directly at the console.
 - The console is a powerful tool used mainly for quick references and serious debugging.
- Operators:
 - Arithmetic operators
 - Comparison operators
 - Logical operators
 - Bitwise operators
 - Assignment operators
 - Identity operators
 - Membership operators



Priority

Variables & Numbers (2)



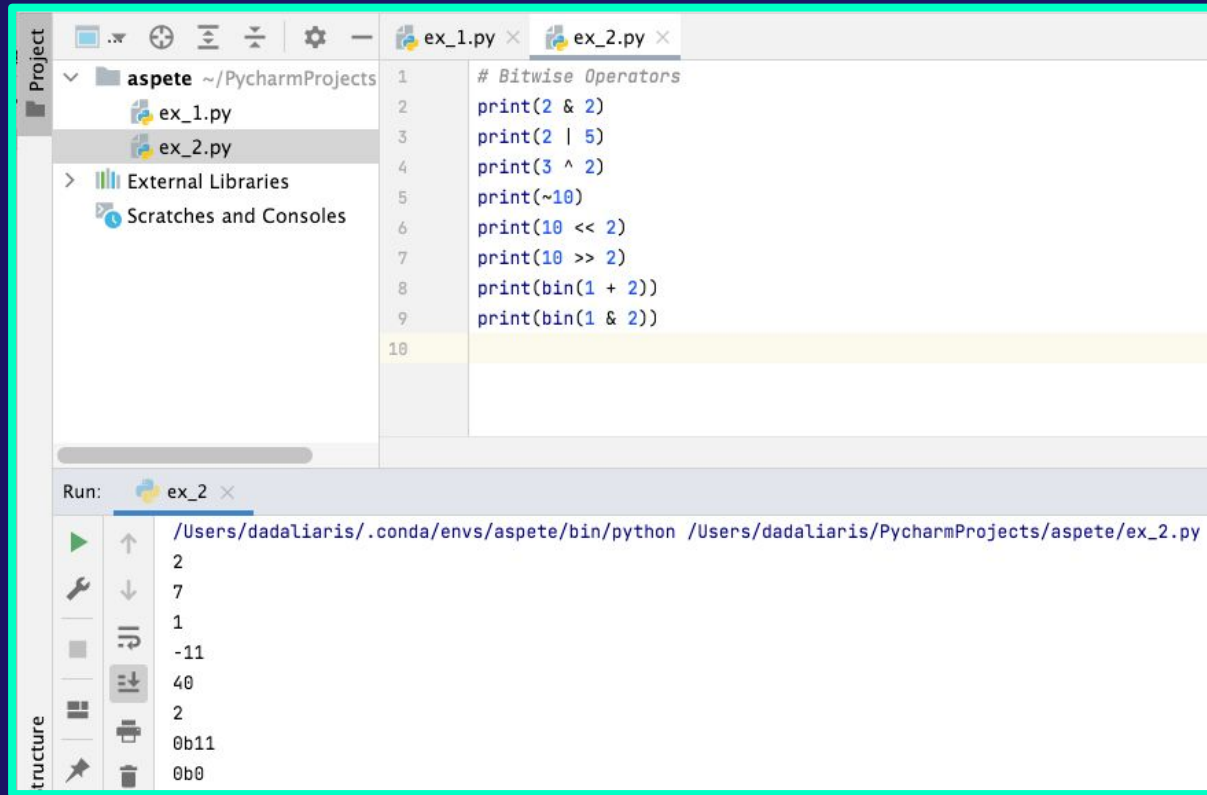
The screenshot displays the PyCharm IDE interface. The main editor window shows a Python file named `ex_1.py` with the following code:

```
1 # Arithmetic Operators
2 print(2 + 2.0)
3 print(2 - 5)
4 print(3 / 4 * 5)
5 print(10 / 10)
6 print(10 / 4)
7 print(10 // 4)
8 print(1 + 2 + 3 + 4 / 4)
9 print((1 + 2 + 3 + 4) / 4)
10 print(2 ** 4)
11 print(7 % 3)
12
```

The Run console at the bottom shows the execution of the script, displaying the following output:

```
Run: ex_1 x
/Users/dadaliliaris/.conda/envs/aspete/bin/python /Users/dadaliliaris/PycharmProjects/aspete/ex_1.py
4.0
-3
3.75
1.0
2.5
2
7.0
2.5
16
1
```

Variables & Numbers (3)



The screenshot displays the PyCharm IDE interface. The top toolbar includes icons for file operations and settings. The Project view on the left shows a project named 'aspete' with files 'ex_1.py' and 'ex_2.py'. The main editor window shows the code for 'ex_2.py' with line numbers 1 through 10. The code performs various bitwise operations. Below the editor, the Run console shows the execution of 'ex_2.py', displaying the output of each print statement.

```
1 # Bitwise Operators
2 print(2 & 2)
3 print(2 | 5)
4 print(3 ^ 2)
5 print(~10)
6 print(10 << 2)
7 print(10 >> 2)
8 print(bin(1 + 2))
9 print(bin(1 & 2))
10
```

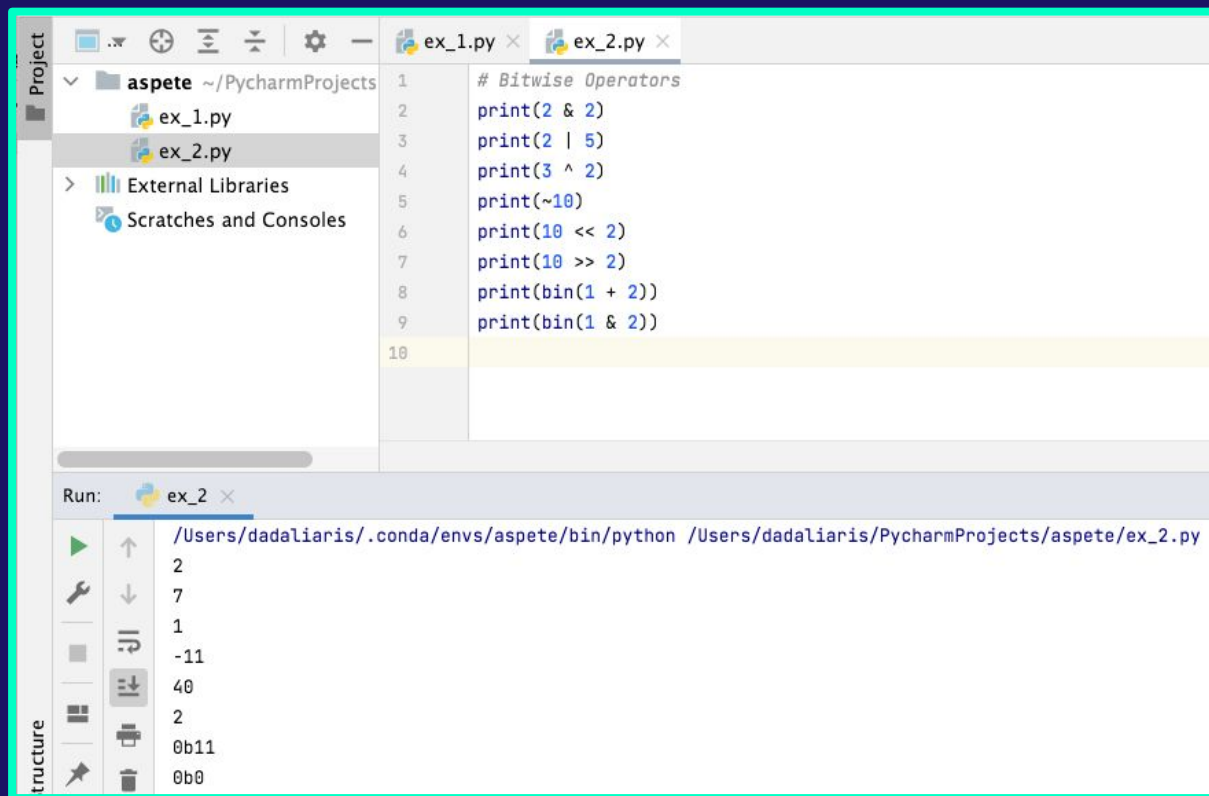
Run: ex_2 ×

```
/Users/dadaljaris/.conda/envs/aspete/bin/python /Users/dadaljaris/PycharmProjects/aspete/ex_2.py
2
7
1
-11
40
2
0b11
0b0
```

Variables & Numbers (4)

- Variable: a reserved memory location to store values.
- Variable Properties:
 - Name
 - Naming Rules: case-sensitive
 - names should always begin with a letter
 - letters, numbers, underscores
 - special symbols(*, -, /, #,)
 - reserved words:
 - False, None, True, and, as, class, continue, def, del, elif, finally, for, from, global, if, is, lambda, nonlocal, not, or, return, try, while, with, yield, import, else, assert, except, in, pass, break, raise
 - Value: various available data types
 - Integers: 1, 10, 232, 34534,
 - Floats: 3.14, 100.12345
 - Complex: 10 + 5j
 - Boolean: True, 1, False, 0
 - List: [0, 1.0, "two"], [1, 2, [3, 4, 5]]
 - Set: {1, 2, 3, "4"}
 - Tuple: (1, 2, "three")
 - Dictionary: {1: "Adam", 2: "Beatris"}

Variables & Numbers (5)



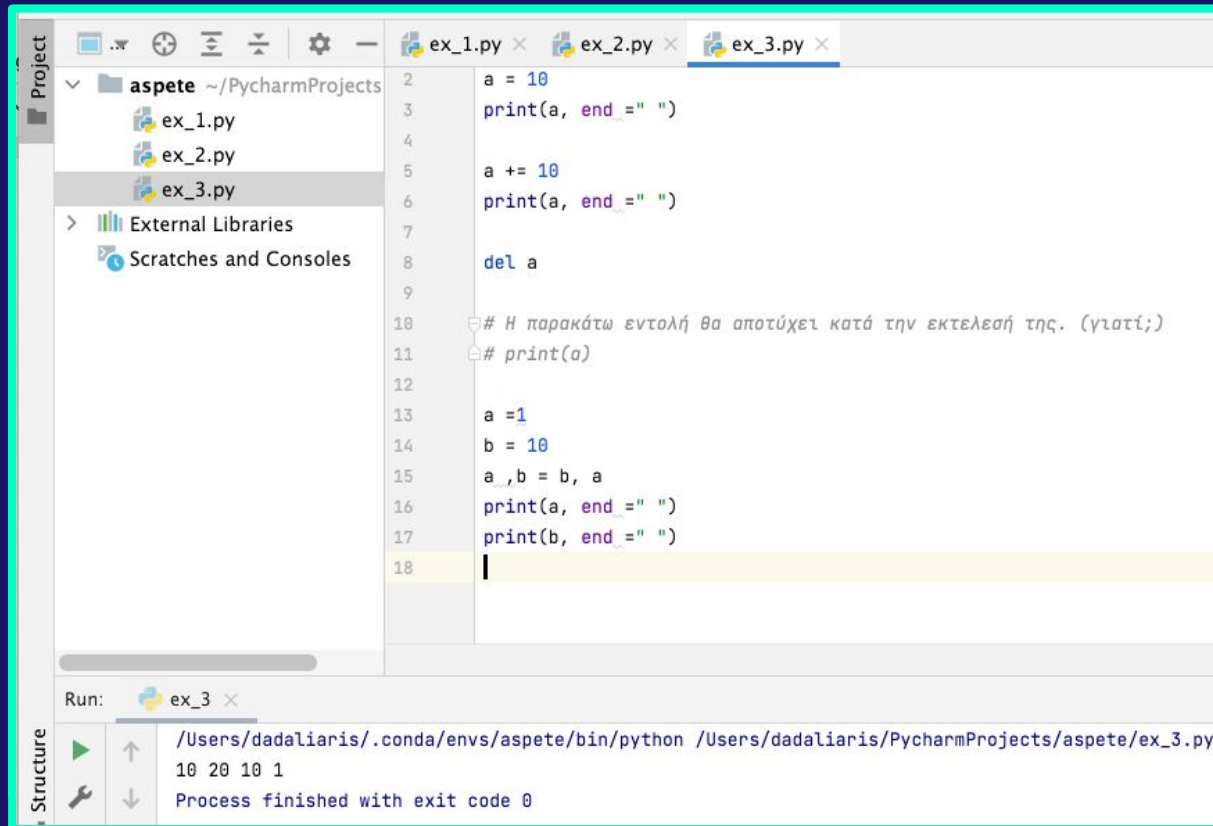
The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for file operations, settings, and window management. The Project view on the left shows a project named 'aspete' with files 'ex_1.py' and 'ex_2.py'. The main editor displays the code for 'ex_2.py' with line numbers 1 through 10. The code uses bitwise operators: &, |, ^, ~, <<, >>, bin(), and &. The Run view at the bottom shows the execution of 'ex_2.py' with the following output:

```
1 # Bitwise Operators
2 print(2 & 2)
3 print(2 | 5)
4 print(3 ^ 2)
5 print(~10)
6 print(10 << 2)
7 print(10 >> 2)
8 print(bin(1 + 2))
9 print(bin(1 & 2))
10
```

Run: ex_2 ×

```
/Users/dadaliliaris/.conda/envs/aspete/bin/python /Users/dadaliliaris/PycharmProjects/aspete/ex_2.py
2
7
1
-11
40
2
0b11
0b0
```

Variables & Numbers (6)



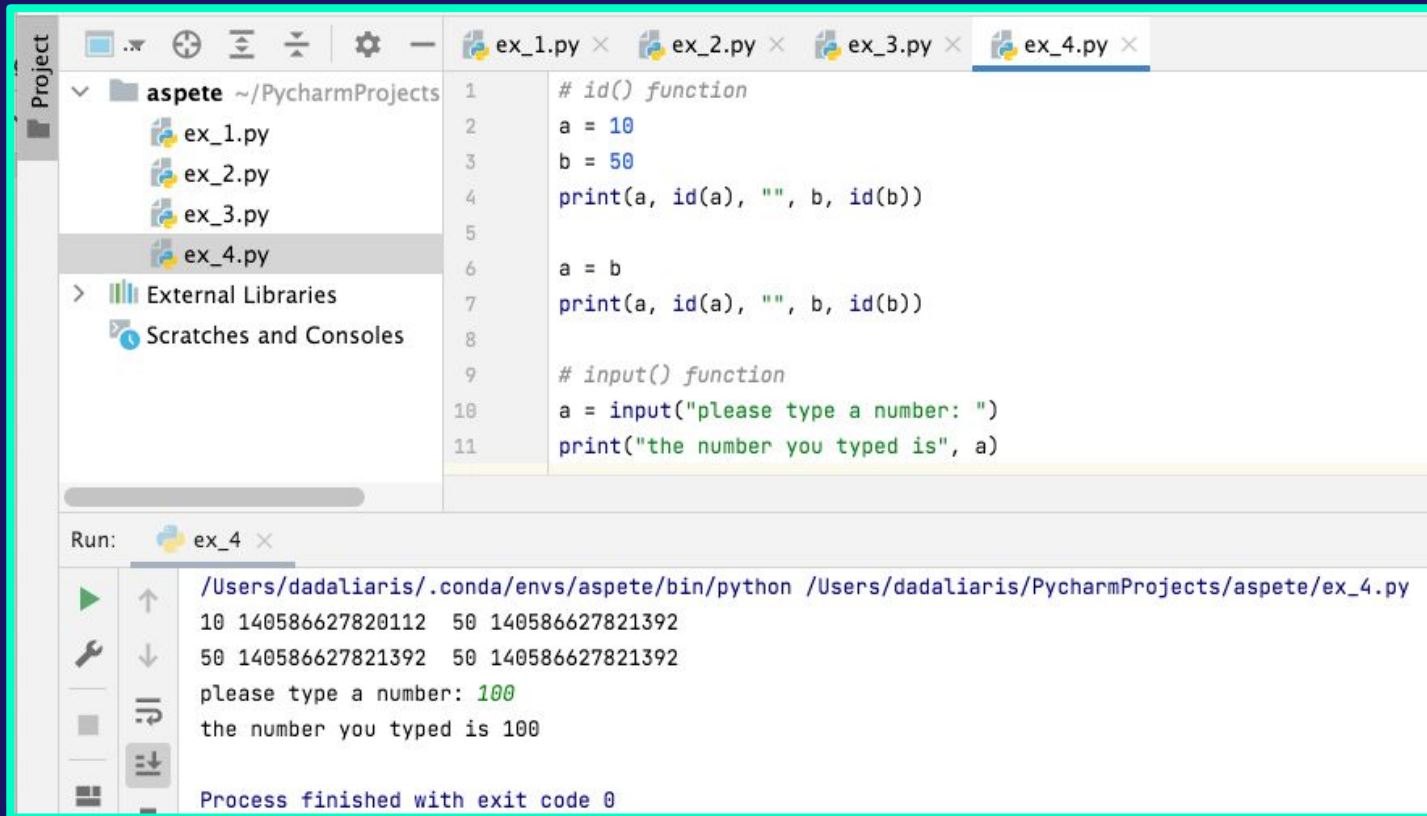
The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure for 'aspete' with files 'ex_1.py', 'ex_2.py', and 'ex_3.py'. The main editor window shows the code in 'ex_3.py' with line numbers 2 through 18. The code defines a variable 'a', prints it, increments it, prints it again, deletes it, and then defines two new variables 'a' and 'b' before printing them. The bottom panel shows the 'Run' output for 'ex_3', displaying the execution path and the output '10 20 10 1', followed by the message 'Process finished with exit code 0'.

```
2 a = 10
3 print(a, end=" ")
4
5 a += 10
6 print(a, end=" ")
7
8 del a
9
10 # Η παρακάτω εντολή θα αποτύχει κατά την εκτέλεσή της. (γιατί;)
11 # print(a)
12
13 a = 1
14 b = 10
15 a, b = b, a
16 print(a, end=" ")
17 print(b, end=" ")
18
```

Run: ex_3 ×

/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_3.py
10 20 10 1
Process finished with exit code 0

Variables & Numbers (7)



The screenshot shows the PyCharm IDE interface. The top toolbar contains icons for file operations and settings. The project view on the left shows a folder named 'aspete' with sub-files 'ex_1.py', 'ex_2.py', 'ex_3.py', and 'ex_4.py'. The main editor window displays the code for 'ex_4.py' with line numbers 1 through 11. The code defines an 'id()' function, assigns values to 'a' and 'b', prints their IDs, reassigns 'a' to 'b', prints the new IDs, and then defines an 'input()' function that prompts for a number and prints it.

```
1  # id() function
2  a = 10
3  b = 50
4  print(a, id(a), "", b, id(b))
5
6  a = b
7  print(a, id(a), "", b, id(b))
8
9  # input() function
10 a = input("please type a number: ")
11 print("the number you typed is", a)
```

Below the editor, the 'Run' panel shows the execution of 'ex_4.py'. The command line is: `/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_4.py`. The output is: `10 140586627820112 50 140586627821392`, `50 140586627821392 50 140586627821392`, `please type a number: 100`, and `the number you typed is 100`. The process finished with exit code 0.

Variables & Numbers (8)

- Data Type Conversions:
 - `int(a, base=n)`
 - `float()`
 - `complex(a b)`
 - `str()`
 - `eval(text)`
- `dir(__builtins__)`:
 - Errors
 - Warnings
 - Interrupts
 - Keywords ('True', 'False', ...)
 - Functions ('abs', 'all', 'any', 'eval', 'id', 'input', 'len', 'max', 'min', ...)

Variables & Numbers (9)

The screenshot displays the PyCharm IDE interface. The top toolbar includes icons for file operations and settings. The project view on the left shows a folder named 'aspete' containing five Python files: ex_1.py, ex_2.py, ex_3.py, ex_4.py, and ex_5.py. The main editor window shows the code for ex_5.py, which is titled '# Data Type Conversions'. The code performs several operations: converting a string to an integer, a negative string to an integer, a string to a float, a string to a string, and evaluating a string to print 'hello'. The Run window at the bottom shows the execution of ex_5.py, displaying the output: '100 <class 'int'> -100 <class 'int'> 4.0 <class 'float'> -1234.56 <class 'str'> hello'. The process finished with exit code 0.

```
1 # Data Type Conversions
2 a = int('100')
3 print(a, type(a), end="\t")
4
5 b = int('-100')
6 print(b, type(b), end="\t")
7
8 c = float('4')
9 print(c, type(c), end="\t")
10
11 d = str('-1234.56')
12 print(d, type(d))
13
14 e = eval('print("hello")')
15 e
```

Run: ex_5 ×

```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_5.py
100 <class 'int'> -100 <class 'int'> 4.0 <class 'float'> -1234.56 <class 'str'>
hello
Process finished with exit code 0
```

Variables & Numbers (10)

The screenshot displays the PyCharm IDE interface. The top toolbar includes icons for file operations and settings. The project view on the left shows a folder named 'aspete' containing files 'ex_1.py' through 'ex_6.py'. The main editor window shows the code for 'ex_6.py' with line numbers 1 through 14. The code uses various built-in Python functions: `abs()`, `bool()`, `max()`, `min()`, `sum()`, and `round()`. The bottom panel shows the execution output for 'ex_6.py', displaying the results of each print statement.

```
1 # Built-in Functions
2 print(abs(-10), end=" ")
3 print(abs(-10.55))
4
5 print(bool(-125), end=" ")
6 print(bool(0))
7
8 print(max(125, -25.6, 22, 1.2), end=" ")
9 print(min(125, -25.6, 22, 1.2), end=" ")
10 print(sum([125, -25.6, 22, 1.2]))
11
12 print(round(1234.56), end=" ")
13 print(round(1234.56, 2))
14
```

Run: ex_6 ×

```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_6.py
10 10.55
True False
125 -25.6 122.60000000000001
1235 1234.56
Process finished with exit code 0
```

Variables & Numbers (11)

- Built-in Libraries:
 - math
 - `acos`, `acosh`, `asin`, `asinh`, `atan`, `atan2`, `ceil`, `copysign`, `cos`, `cosh`, `degrees`, `e`, `erf`, `erfc`, `exp`, `expm1`, `fabs`, `factorial`, `floor`, `fmod`, `frexp`, `fsum`, `gamma`, `gcd`, `hypot`, `inf`, `isclose`, `isfinite`, `isinf`, `isnan`, `ldexp`, `lgamma`, `log`, `log10`, `loglp`, `log2`, `modf`, `nan`, `pi`, `pow`, `radians`, `sin`, `sinh`, `sqrt`, `tan`, `tanh`, `tau`, `trunc`
 - random
 - `random.seed(number)`
 - `random.seed()` (uses the system clock by default)
 - `random.randint(from_a, to_b)`
 - `random.random()`
 -
- “External” Libraries - Modules:
 - Fractions
 - pandas
 - tkinter
 -

Flow Control / Conditional Statements (1)

- It all comes down to booleans
 - Values:
 - True (1)
 - False (0)
 - Boolean numbers can be used in classic arithmetic operations:
 - Example: $a = \text{True} + 2 \rightarrow a=3$

<code>==</code>	<code>1 == 2</code>	False
<code>!=</code>	<code>1 != 2</code>	True
<code>></code>	<code>5 > 10</code>	False
<code><</code>	<code>5 < 10</code>	True
<code>>=</code>	<code>4 >= 4</code>	True
<code><=</code>	<code>4 <= 10</code>	True

Flow Control / Conditional Statements (2)

- Logical Operators:
 - and
 - or
 - not

a	b	a and b	a or b	not(a)
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Flow Control / Conditional Statements (3)

- Operators:
 - lambda
 - If-else
 - or
 - and
 - not
 - in, not in, is, is not, <, <=, >, >=, !=, ==
 - |
 - ^
 - &
 - <<, >>
 - +, -
 - *, /, //, %
 - **
 - ()

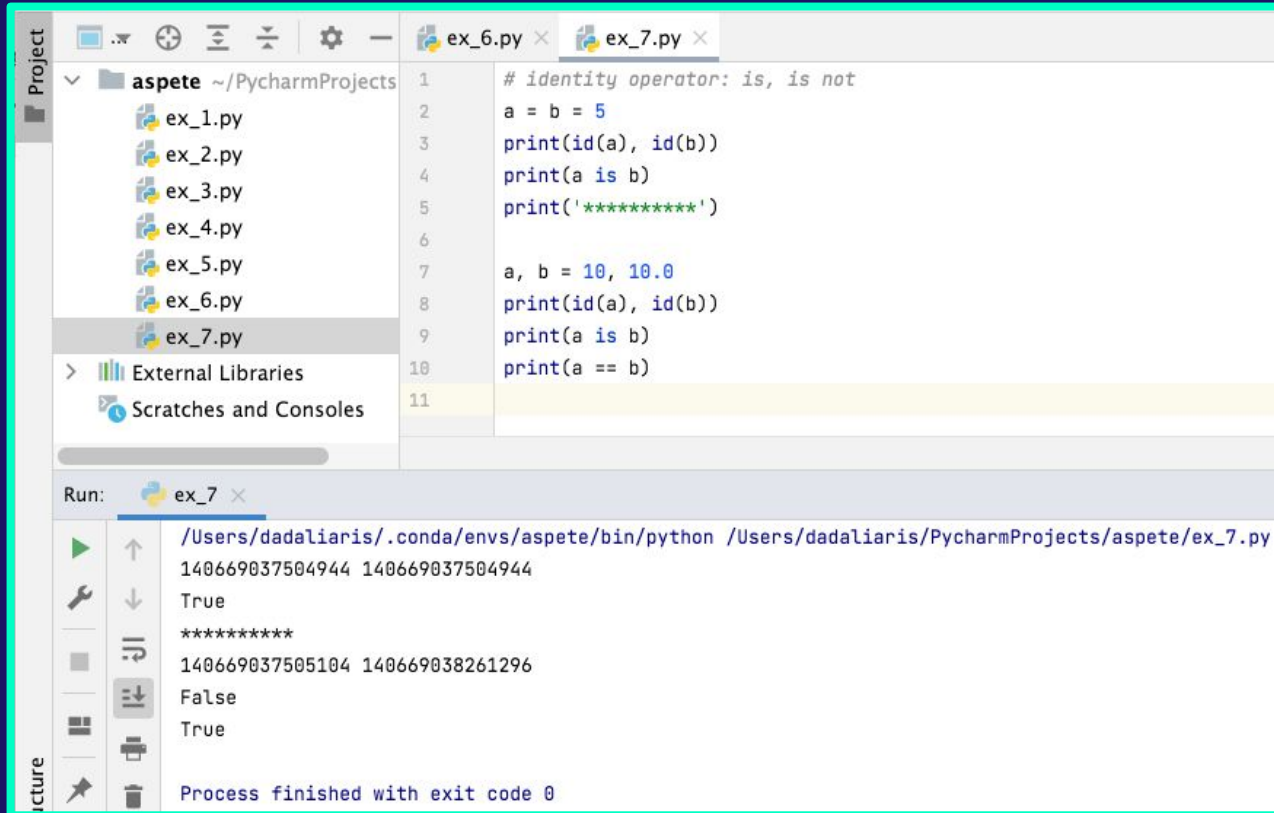
Priority



Chain comparison:

- $a < b < c$
- $a < b > c$
- $a > b < c$

Flow Control / Conditional Statements (4)



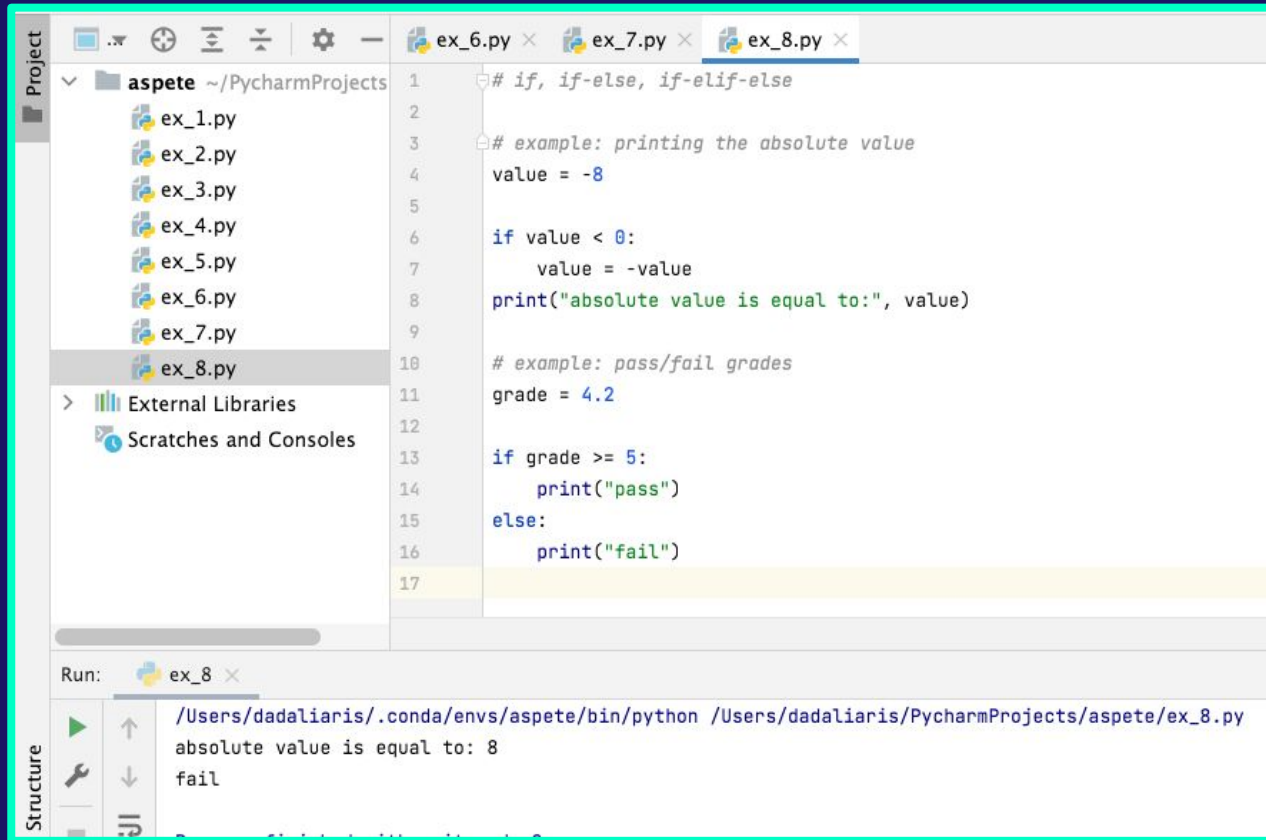
The screenshot shows the PyCharm IDE interface. The left sidebar displays a project named 'aspete' with files 'ex_1.py' through 'ex_7.py'. The main editor window shows the code for 'ex_7.py' with line numbers 1 through 11. The code defines two variables, 'a' and 'b', and prints their IDs, the result of the identity operator 'is', and the result of the equality operator '=='.

```
1 # identity operator: is, is not
2 a = b = 5
3 print(id(a), id(b))
4 print(a is b)
5 print('*****')
6
7 a, b = 10, 10.0
8 print(id(a), id(b))
9 print(a is b)
10 print(a == b)
11
```

Below the editor, the 'Run' panel shows the execution of 'ex_7.py'. The command used is: `/Users/dadalialis/.conda/envs/aspete/bin/python /Users/dadalialis/PycharmProjects/aspete/ex_7.py`. The output is as follows:

```
140669037504944 140669037504944
True
*****
140669037505104 140669038261296
False
True
Process finished with exit code 0
```

Flow Control / Conditional Statements (5)



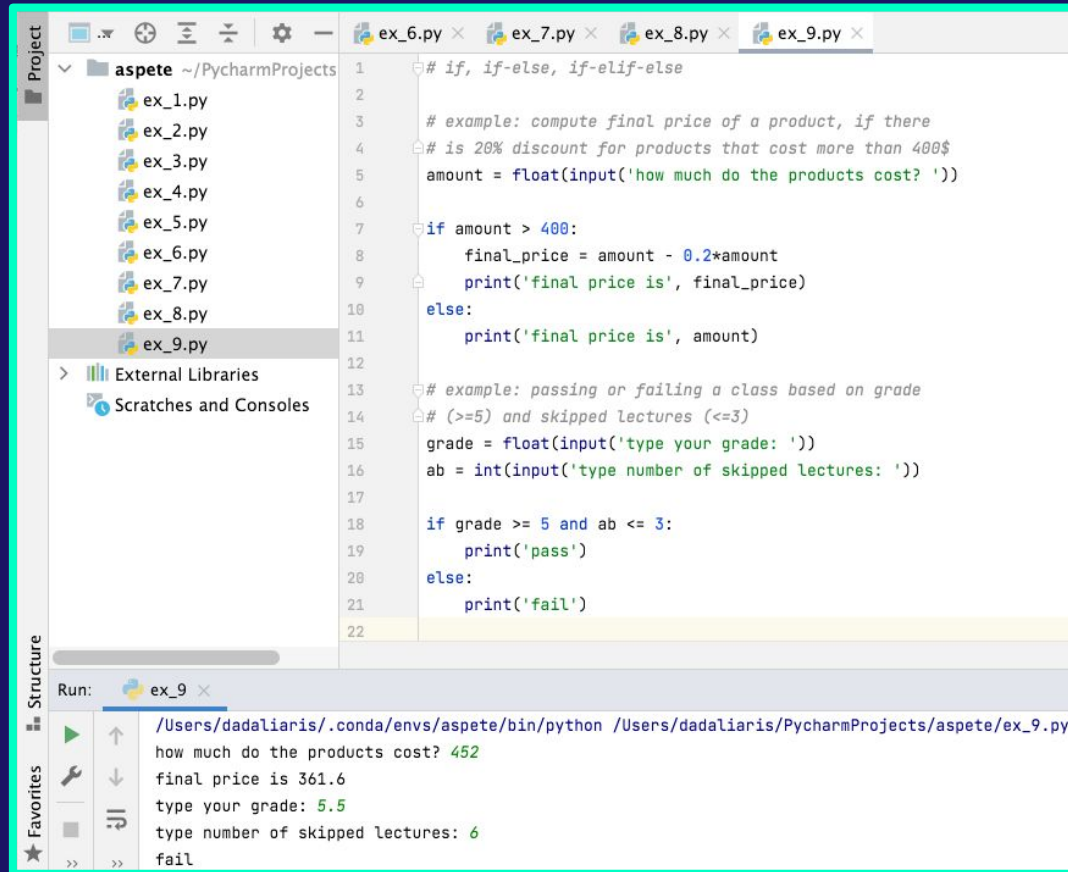
The screenshot displays the PyCharm IDE interface. The left sidebar shows a project named 'aspete' with files 'ex_1.py' through 'ex_8.py'. The main editor window shows the code for 'ex_8.py' with line numbers 1 through 17. The code includes comments and two conditional blocks. The first block checks if a value is less than 0 and prints its absolute value. The second block checks if a grade is greater than or equal to 5 and prints 'pass' or 'fail'. The bottom panel shows the output of running 'ex_8.py', which is 'absolute value is equal to: 8' followed by 'fail' on a new line.

```
1  # if, if-else, if-elif-else
2
3  # example: printing the absolute value
4  value = -8
5
6  if value < 0:
7      value = -value
8  print("absolute value is equal to:", value)
9
10 # example: pass/fail grades
11 grade = 4.2
12
13 if grade >= 5:
14     print("pass")
15 else:
16     print("fail")
17
```

Run: ex_8 ×

```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_8.py
absolute value is equal to: 8
fail
```


Flow Control / Conditional Statements (6)



The screenshot displays the PyCharm IDE interface. The left sidebar shows a project named 'aspete' with files 'ex_1.py' through 'ex_9.py'. The main editor window shows the code for 'ex_9.py'. The code includes comments and logic for calculating a final price based on a discount and determining a pass/fail status based on a grade and the number of skipped lectures. The bottom panel shows the execution output for 'ex_9.py' with user input and program output.

```
1  # if, if-else, if-elif-else
2
3  # example: compute final price of a product, if there
4  # is 20% discount for products that cost more than 400$
5  amount = float(input('how much do the products cost? '))
6
7  if amount > 400:
8      final_price = amount - 0.2*amount
9      print('final price is', final_price)
10 else:
11     print('final price is', amount)
12
13 # example: passing or failing a class based on grade
14 # (>=5) and skipped lectures (<=3)
15 grade = float(input('type your grade: '))
16 ab = int(input('type number of skipped lectures: '))
17
18 if grade >= 5 and ab <= 3:
19     print('pass')
20 else:
21     print('fail')
22
```

Run: ex_9 ×

```
/Users/dadalariis/.conda/envs/aspete/bin/python /Users/dadalariis/PycharmProjects/aspete/ex_9.py
how much do the products cost? 452
final price is 361.6
type your grade: 5.5
type number of skipped lectures: 6
fail
```

Flow Control / Conditional Statements (7)

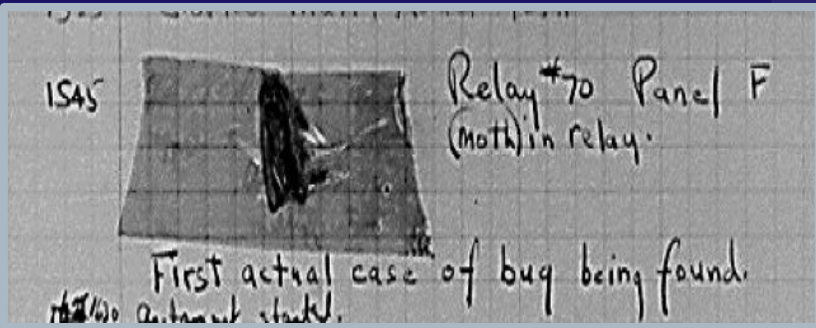
The screenshot displays the PyCharm IDE interface. On the left, the Project tool window shows a folder named 'aspete' containing files 'ex_1.py' through 'ex_10.py'. The main editor window shows the code for 'ex_10.py' with the following content:

```
1  # if, if-else, if-elif-else
2
3  # example: check whether a number given by the user is positive,
4  # negative or equal to zero, without using build-in functions
5  num = int(input('type an integer: '))
6
7  if num >= 0:
8      if num == 0:
9          print('zero')
10         else:
11             print('positive')
12     else:
13         print('negative')
14
```

At the bottom, the Run tool window shows the execution of 'ex_10.py' with the following output:

```
Run: ex_10 x
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_10.py
type an integer: -15
negative
Process finished with exit code 0
```

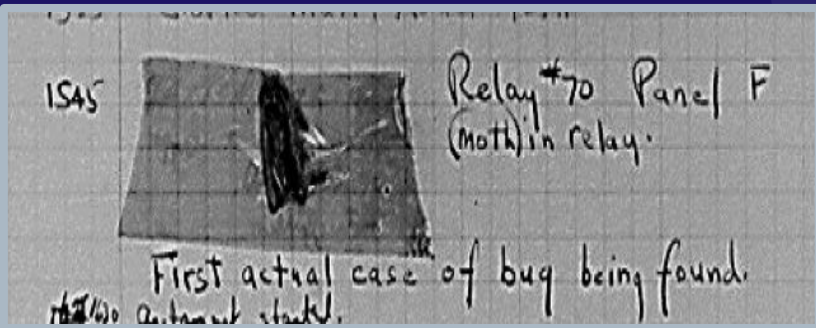
Debugging (1)



- Critical skill in programming (you must learn how to do it and do it well)
- No single way to do it
- **Biggest obstacle: “my code is right and the computer is wrong”**
- Your best “friend”: the debugger
- Two approaches depending on the program’s misbehavior:
 - Program runs without crashing: use the debugger to watch the program’s execution, stepping through the program one statement at a time.
 - Program is crashing: start program execution under the debugger, the debugger will automatically get control at the point of the crash, where you can examine variable values at that point.
- Why is it called a “bug”: In the 1940s, in a U.S. Navy computer project using one of the first general-purpose computers (Mark I), a programmer found that a problem was due to a dead moth stuck in a relay in the machine. Removing the bug allowed the computer to run as programmed.

Debugging (2)

- “pass” statement
 - a null operation, when it is executed, nothing happens
 - useful as a placeholder when a statement is required syntactically, but no code needs to be executed
- “assert” statement:
 - statement has a condition or expression which is supposed to be always true. If the condition is false assert halts the program and gives an `AssertionError`
- Exceptions:
 - More on this later. . .



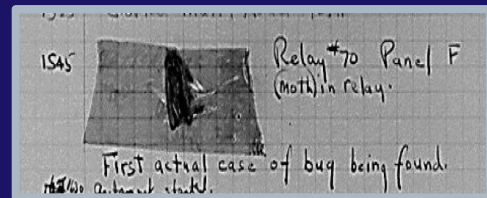
Debugging (3)

The screenshot shows the PyCharm IDE with a project containing several Python files (ex_1.py to ex_11.py). The file ex_11.py is open and contains the following code:

```
1 import math
2
3 # pass statement
4 x = float(input('type a number: '))
5
6 if x >= 0:
7     pass
8 else:
9     x = x + 5
10
11 # assert statement
12 assert x >= 0
13 squared_x = math.sqrt(x)
14
```

The Run console shows the execution of ex_11.py with the following output:

```
Run: ex_11 x
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_11.py
type a number: -11
Traceback (most recent call last):
  File "/Users/dadaliaris/PycharmProjects/aspete/ex_11.py", line 12, in <module>
    assert x >= 0
AssertionError
Process finished with exit code 1
```



Python Loops (1)

- Python has two basic loop constructs:
 - for
 - while
- Before we check their syntax and usage we should mention the “range” built-in function and the “in” member operator.
- “range” is used to produce a sequence of numbers
- range(start, finish, step)
- “in” is used for checking the existence of a value in a sequence of values

Python Loops (2)

The screenshot shows a Python IDE with a project named 'Project' containing several Python files (ex_2.py to ex_12.py). The file 'ex_12.py' is open and contains the following code:

```
1 # range(), built-in function
2 print(list(range(10)))
3 print(list(range(4, 10)))
4 print(list(range(2, 10, 3)))
5
6 my_list = list(range(5))
7 print(my_list)
8
9 # in, member operator
10 print(7 in [1, 2, 3, 4, 5])
11 print(7 not in [1, 2, 3, 4, 5])
12 print('p' in 'Python')
13 print('Py' in 'Python')
14
```

The 'Run' console shows the output for 'ex_12.py':

```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_12.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[4, 5, 6, 7, 8, 9]
[2, 5, 8]
[0, 1, 2, 3, 4]
False
True
False
True
```

Python Loops (3)

The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure for 'aspete' with files 'ex_1.py' through 'ex_13.py'. The main editor window shows the code for 'ex_13.py' with line numbers 1 through 16. The code includes three examples of loops: a range loop, a loop over a list of artists, and a loop over a list of bands. The bottom panel shows the output of the code execution, including the range of integers and the formatted output for the artists and bands.

```
1 # example: a simple way to print the 10 first integers
2 for i in range(10):
3     print(i, end=',')
4 print('\n')
5
6 # example:
7 artists = ['Reznor', 'Vedder', 'Keenan']
8
9 for artist in artists:
10     print(artist, end=',')
11
12 # example: the non-pythonic way
13 bands = ['NIN', 'Pearl Jam', 'Tool']
14
15 for i in range(len(artists)):
16     print(artists[i], '-', bands[i])
```

Run: ex_13 ×

/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/
0,1,2,3,4,5,6,7,8,9,

Reznor,Vedder,Keenan,Reznor - NIN
Vedder - Pearl Jam
Keenan - Tool

Python Loops (4)

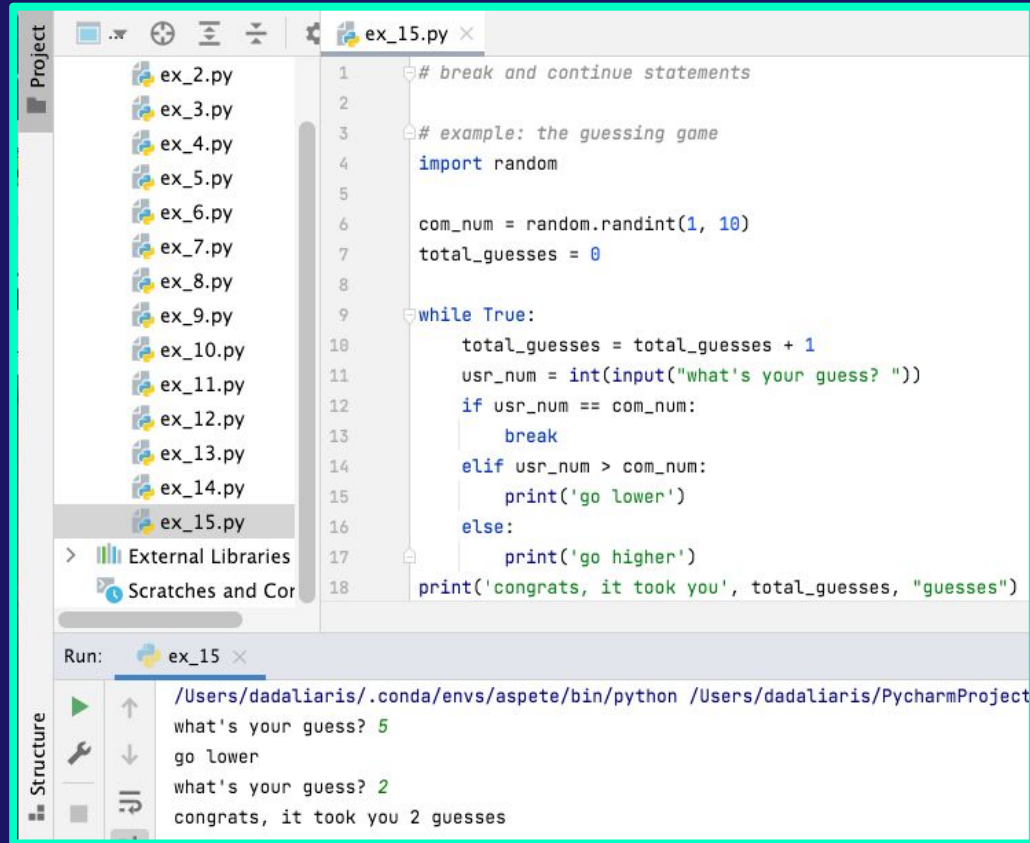
The screenshot shows a Python IDE with a project named 'Project' containing files from ex_3.py to ex_14.py. The editor displays the following code:

```
1 # example: zip() function
2 list_1 = [1, 2, 3, 4]
3 list_2 = ['one', 'two', 'three', 'four']
4
5 for dig, nam in zip(list_1, list_2):
6     print(dig, nam)
7
8 # example: calculate the power of two for each given
9 # number, until a negative number is given
10 number = float(input('type a number: '))
11
12 while number >= 0:
13     print(number, 'squared is equal to', number*number)
14     number = float(input('type a number: '))
```

The Run console shows the execution of ex_14.py:

```
Run: ex_14 x
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_14.py
1 one
2 two
3 three
4 four
type a number: 4
4.0 squared is equal to 16.0
type a number: -2
```

Python Loops (5)



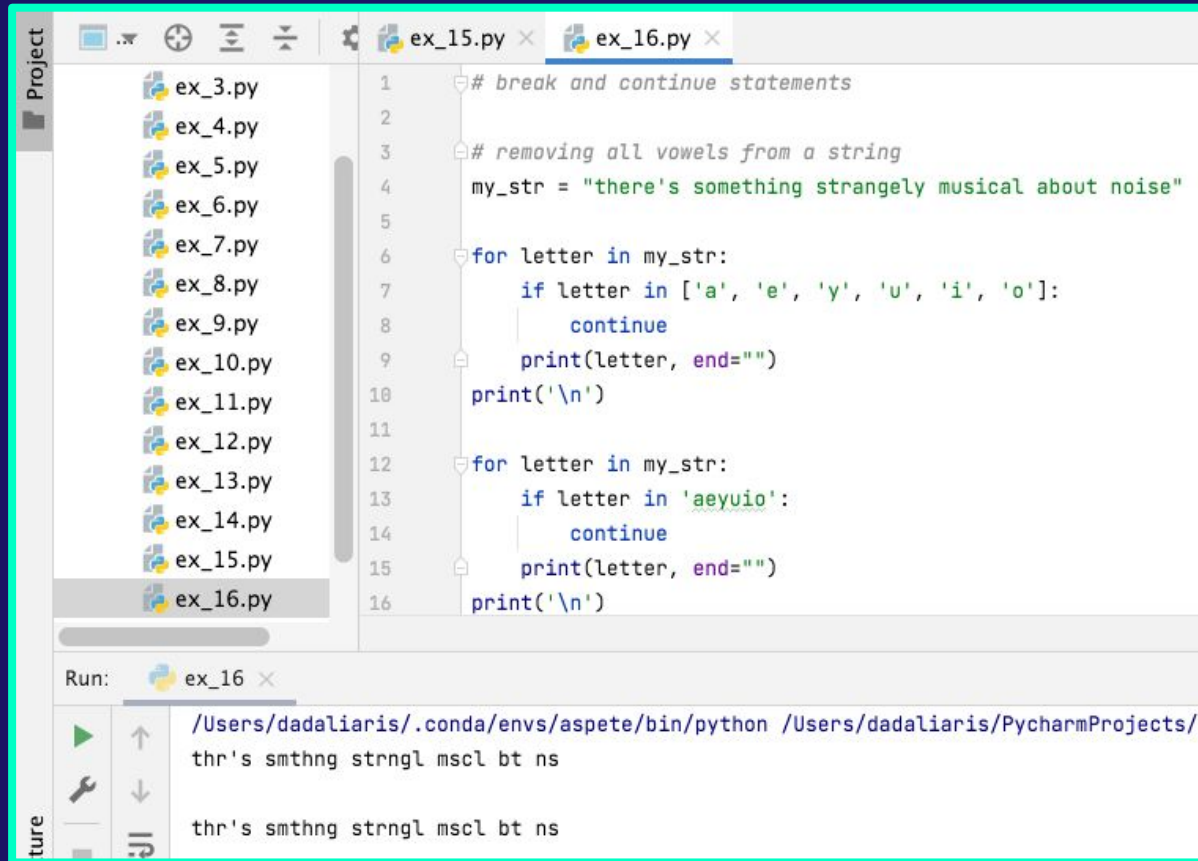
The screenshot shows a Python IDE with a project named 'PycharmProject'. The file explorer on the left lists files from ex_2.py to ex_15.py. The main editor displays the code for ex_15.py, which is a guessing game. The code uses a while loop to repeatedly prompt the user for a guess until it matches the computer's random number. The output window at the bottom shows the program's execution, including the user's input and the program's feedback.

```
1  # break and continue statements
2
3  # example: the guessing game
4  import random
5
6  com_num = random.randint(1, 10)
7  total_guesses = 0
8
9  while True:
10     total_guesses = total_guesses + 1
11     usr_num = int(input("what's your guess? "))
12     if usr_num == com_num:
13         break
14     elif usr_num > com_num:
15         print('go lower')
16     else:
17         print('go higher')
18     print('congrats, it took you', total_guesses, "guesses")
```

Run: ex_15

```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProject
what's your guess? 5
go lower
what's your guess? 2
congrats, it took you 2 guesses
```

Python Loops (6)



The screenshot shows a Python IDE with a project explorer on the left containing files from ex_3.py to ex_16.py. The main editor displays the code for ex_16.py, which demonstrates the use of the `continue` statement in a `for` loop. The code defines a string `my_str` and iterates over each character. In the first loop, characters 'a', 'e', 'y', 'u', 'i', and 'o' are skipped. In the second loop, characters 'a', 'e', 'y', 'u', 'i', and 'o' are also skipped, leaving only 'h', 'r', 's', ' ', 's', 'm', 't', 'h', 'n', 'g', ' ', 's', 't', 'r', 'n', 'g', 'l', ' ', 'm', 's', 'c', 'l', ' ', 'b', 't', ' ', 'n', 's' to be printed.

```
1  # break and continue statements
2
3  # removing all vowels from a string
4  my_str = "there's something strangely musical about noise"
5
6  for letter in my_str:
7      if letter in ['a', 'e', 'y', 'u', 'i', 'o']:
8          continue
9      print(letter, end="")
10     print('\n')
11
12     for letter in my_str:
13         if letter in 'aeyuio':
14             continue
15         print(letter, end="")
16     print('\n')
```

Run: ex_16 x

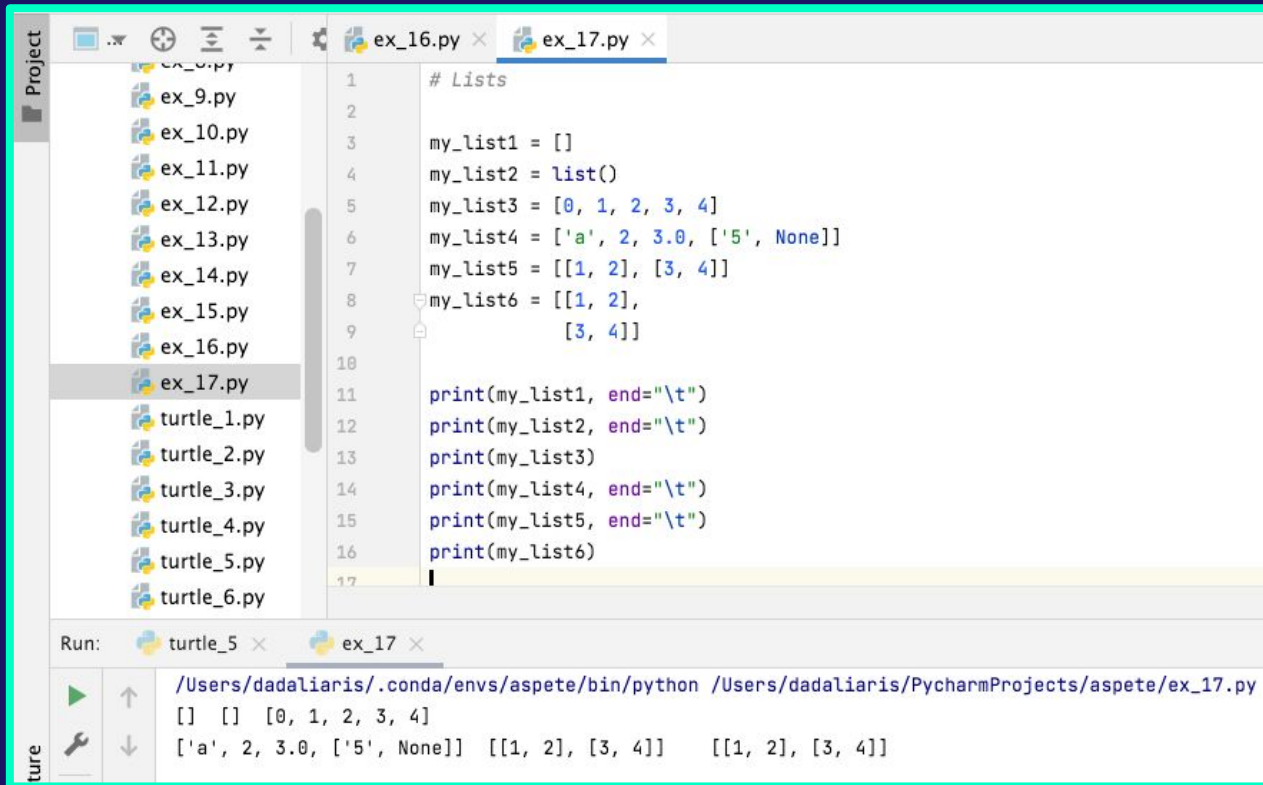
```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/
thr's smthng strngl mscl bt ns

thr's smthng strngl mscl bt ns
```

Python Lists (1)

- List Definition & Characteristics:
 - A collection or sequence of **ordered** objects.
 - Dynamic, its size can change at any time during the execution of a program.
 - Can contain objects of any size and any type.
 - Has built-in methods.
 - Is a mutable object that might contain immutable elements.

Python Lists (2)



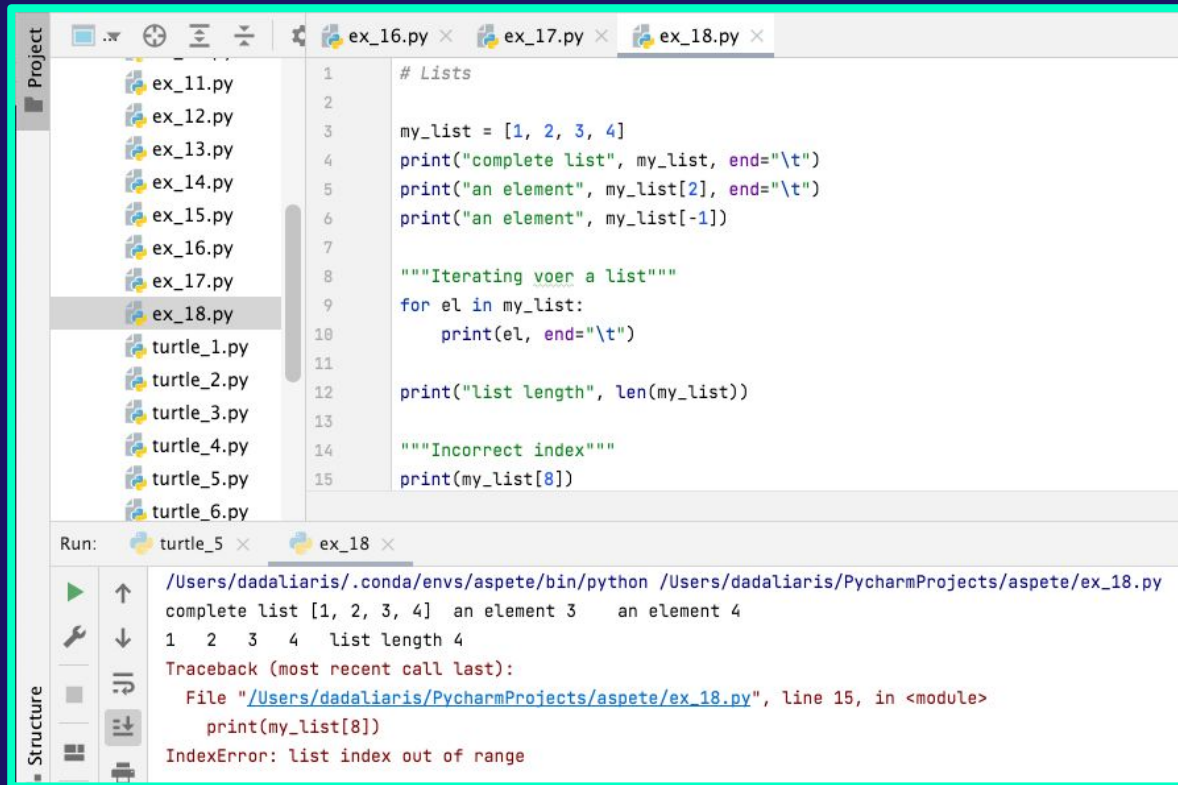
The screenshot shows a Python IDE with a project view on the left and a code editor on the right. The project view lists files from ex_8.py to turtle_6.py. The code editor shows the contents of ex_17.py, which defines six lists and prints them. The output window at the bottom shows the execution results for ex_17.py.

```
1 # Lists
2
3 my_list1 = []
4 my_list2 = list()
5 my_list3 = [0, 1, 2, 3, 4]
6 my_list4 = ['a', 2, 3.0, ['5', None]]
7 my_list5 = [[1, 2], [3, 4]]
8 my_list6 = [[1, 2],
9             [3, 4]]
10
11 print(my_list1, end="\t")
12 print(my_list2, end="\t")
13 print(my_list3)
14 print(my_list4, end="\t")
15 print(my_list5, end="\t")
16 print(my_list6)
17
```

Run: turtle_5 × ex_17 ×

```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_17.py
[] [] [0, 1, 2, 3, 4]
['a', 2, 3.0, ['5', None]] [[1, 2], [3, 4]] [[1, 2], [3, 4]]
```

Python Lists (3)



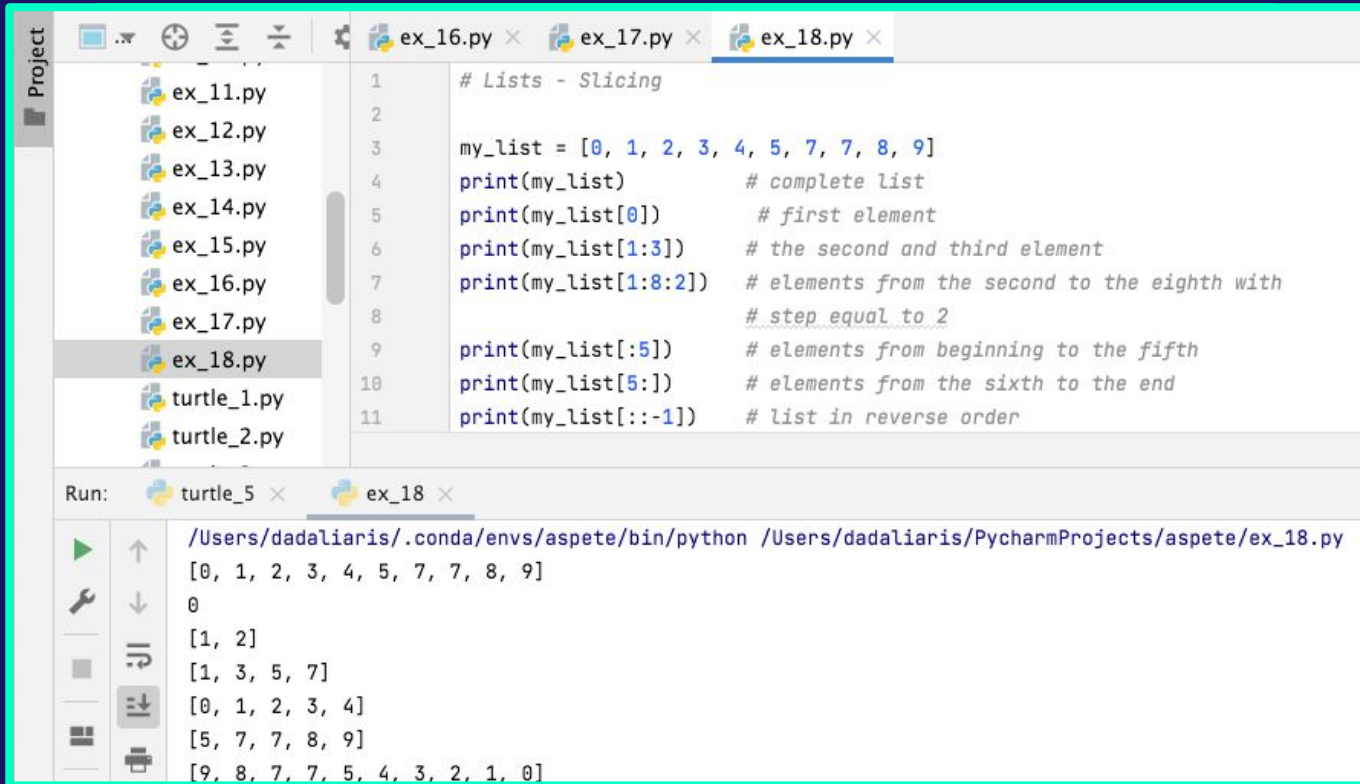
The screenshot shows a PyCharm IDE window with three tabs: ex_16.py, ex_17.py, and ex_18.py. The editor displays the following Python code:

```
1 # Lists
2
3 my_list = [1, 2, 3, 4]
4 print("complete list", my_list, end="\t")
5 print("an element", my_list[2], end="\t")
6 print("an element", my_list[-1])
7
8 """Iterating over a list"""
9 for el in my_list:
10     print(el, end="\t")
11
12 print("list length", len(my_list))
13
14 """Incorrect index"""
15 print(my_list[8])
```

The Run console at the bottom shows the output of the code:

```
Run: turtle_5 x ex_18 x
/Users/dadalialis/.conda/envs/aspete/bin/python /Users/dadalialis/PycharmProjects/aspete/ex_18.py
complete list [1, 2, 3, 4] an element 3 an element 4
1 2 3 4 list length 4
Traceback (most recent call last):
  File "/Users/dadalialis/PycharmProjects/aspete/ex_18.py", line 15, in <module>
    print(my_list[8])
IndexError: list index out of range
```

Python Lists (4)



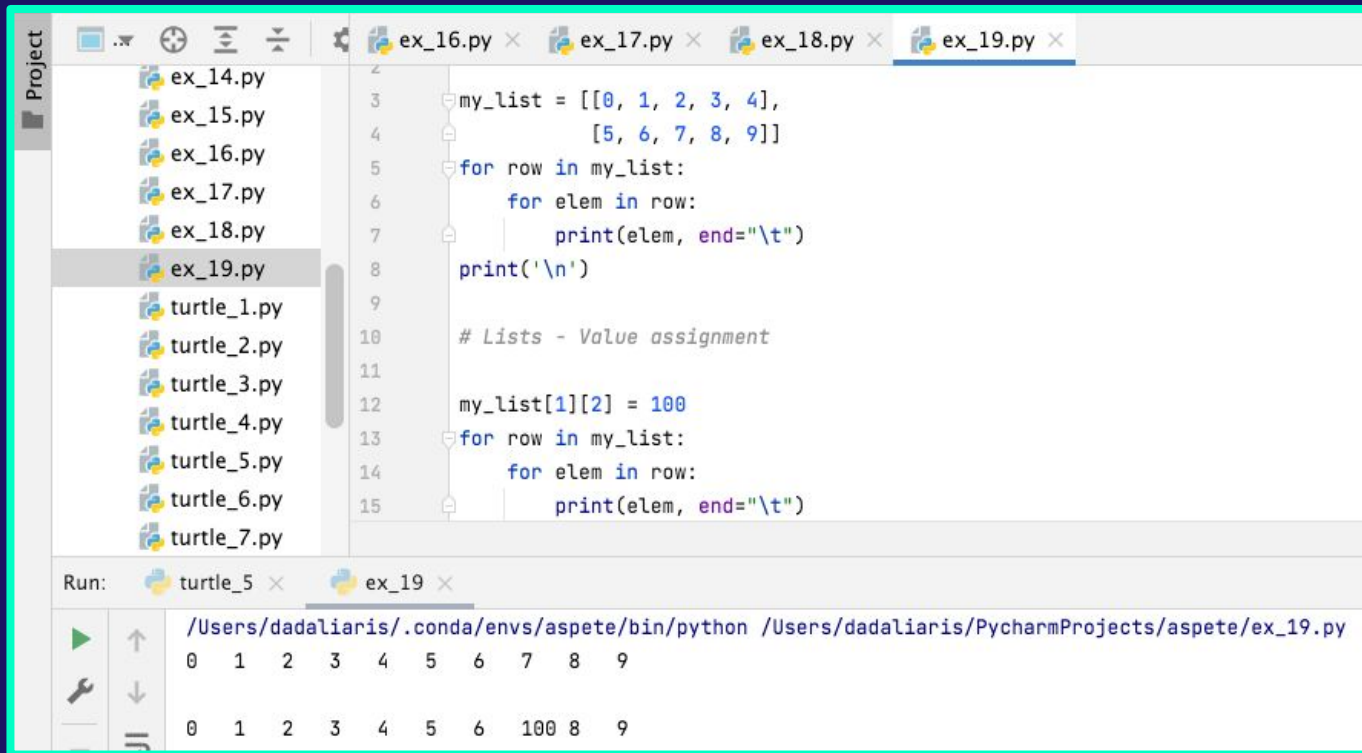
The screenshot shows a Python IDE with a project view on the left and a code editor on the right. The code editor displays a Python script named `ex_18.py` that demonstrates list slicing. The code defines a list `my_list` and prints it using various slicing techniques. The output window at the bottom shows the execution results for `ex_18`.

```
1 # Lists - Slicing
2
3 my_list = [0, 1, 2, 3, 4, 5, 7, 7, 8, 9]
4 print(my_list)           # complete list
5 print(my_list[0])       # first element
6 print(my_list[1:3])     # the second and third element
7 print(my_list[1:8:2])   # elements from the second to the eighth with
8                         # step equal to 2
9 print(my_list[:5])      # elements from beginning to the fifth
10 print(my_list[5:])     # elements from the sixth to the end
11 print(my_list[::-1])   # list in reverse order
```

Run: turtle_5 × ex_18 ×

```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_18.py
[0, 1, 2, 3, 4, 5, 7, 7, 8, 9]
0
[1, 2]
[1, 3, 5, 7]
[0, 1, 2, 3, 4]
[5, 7, 7, 8, 9]
[9, 8, 7, 7, 5, 4, 3, 2, 1, 0]
```

Python Lists (5)



The screenshot shows a Python IDE with a project view on the left and a code editor on the right. The code editor displays the following Python code:

```
2  
3 my_list = [[0, 1, 2, 3, 4],  
4           [5, 6, 7, 8, 9]]  
5 for row in my_list:  
6     for elem in row:  
7         print(elem, end="\t")  
8     print('\n')  
9  
10 # Lists - Value assignment  
11  
12 my_list[1][2] = 100  
13 for row in my_list:  
14     for elem in row:  
15         print(elem, end="\t")
```

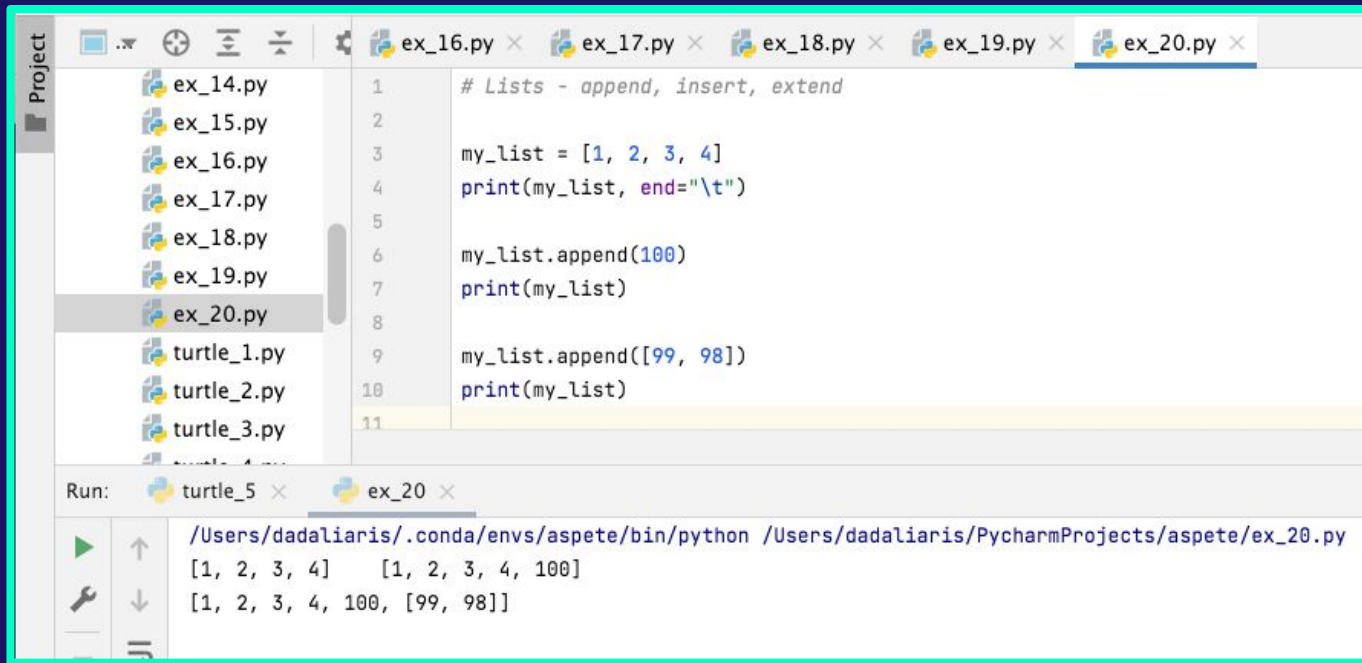
The Run console at the bottom shows the output of the code:

```
Run: turtle_5 x ex_19 x  
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_19.py  
0 1 2 3 4 5 6 7 8 9  
0 1 2 3 4 5 6 100 8 9
```


Python Lists (6)

- There are three distinct ways to add new elements on a list:
 - “append”: adds a new element at the end of a list (in-place)
 - “insert”: adds a new element at a specific index of a list
 - “extend”: adds a new sequence of elements at the end of a list
- There are four distinct ways to delete a list or elements of a list:
 - “del”
 - “clear”
 - “remove”
 - “pop”

Python Lists (7)



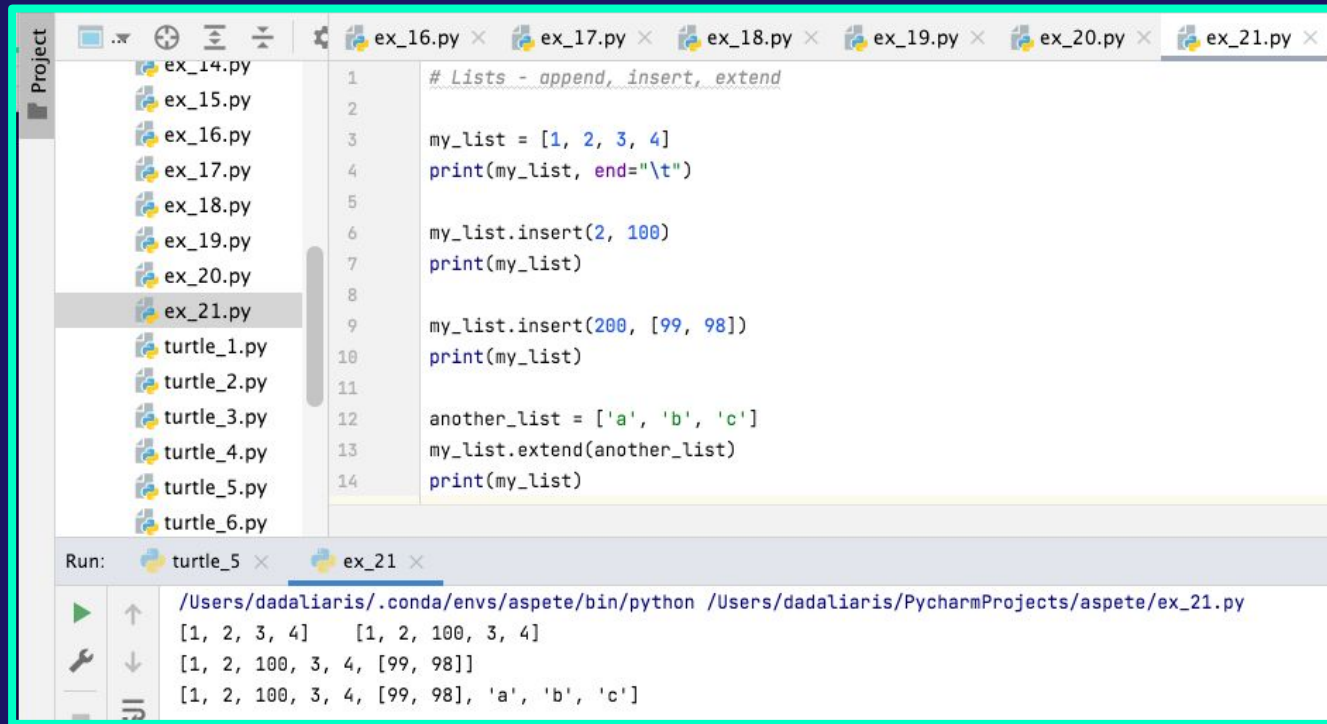
The screenshot shows a Python IDE with a project explorer on the left and a code editor on the right. The project explorer lists files from ex_14.py to turtle_4.py. The code editor shows the following Python code:

```
1 # Lists - append, insert, extend
2
3 my_list = [1, 2, 3, 4]
4 print(my_list, end="\t")
5
6 my_list.append(100)
7 print(my_list)
8
9 my_list.append([99, 98])
10 print(my_list)
11
```

Below the code editor, the Run console shows the output of the script:

```
Run: turtle_5 x ex_20 x
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_20.py
[1, 2, 3, 4] [1, 2, 3, 4, 100]
[1, 2, 3, 4, 100, [99, 98]]
```

Python Lists (8)



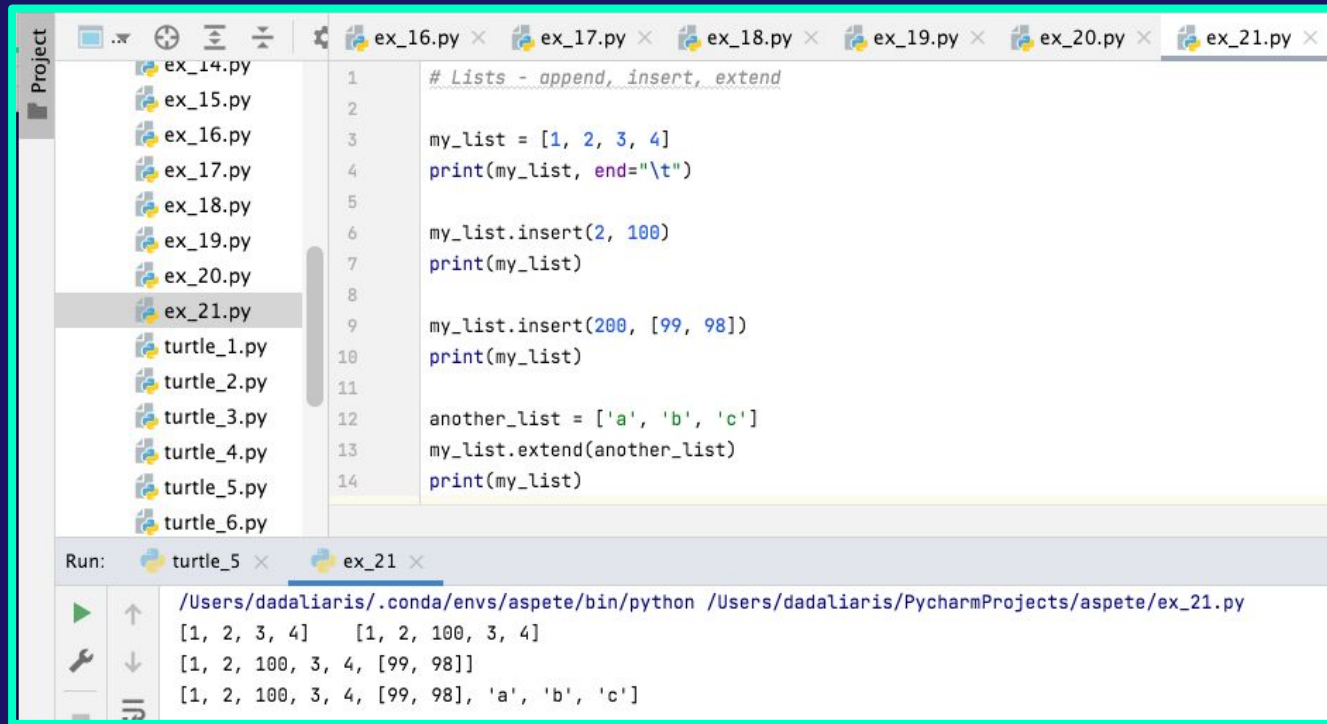
The screenshot shows a Python IDE with a project explorer on the left and a code editor on the right. The project explorer lists files from ex_14.py to turtle_6.py. The code editor shows the following Python code:

```
1  # Lists - append, insert, extend
2
3  my_list = [1, 2, 3, 4]
4  print(my_list, end="\t")
5
6  my_list.insert(2, 100)
7  print(my_list)
8
9  my_list.insert(200, [99, 98])
10 print(my_list)
11
12 another_list = ['a', 'b', 'c']
13 my_list.extend(another_list)
14 print(my_list)
```

At the bottom, a Run console shows the output of the script:

```
Run: turtle_5 x ex_21 x
/Users/dadalIaris/.conda/envs/aspete/bin/python /Users/dadalIaris/PycharmProjects/aspete/ex_21.py
[1, 2, 3, 4] [1, 2, 100, 3, 4]
[1, 2, 100, 3, 4, [99, 98]]
[1, 2, 100, 3, 4, [99, 98], 'a', 'b', 'c']
```

Python Lists (9)



The screenshot shows a Python IDE with a project explorer on the left containing files from ex_14.py to turtle_6.py. The main editor displays the code for ex_21.py, which demonstrates list operations: initialization, printing, inserting an integer, inserting a list, and extending with another list. The Run console at the bottom shows the output of these operations.

```
1  # Lists - append, insert, extend
2
3  my_list = [1, 2, 3, 4]
4  print(my_list, end="\t")
5
6  my_list.insert(2, 100)
7  print(my_list)
8
9  my_list.insert(200, [99, 98])
10 print(my_list)
11
12 another_list = ['a', 'b', 'c']
13 my_list.extend(another_list)
14 print(my_list)
```

Run: turtle_5 × ex_21 ×

```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_21.py
[1, 2, 3, 4] [1, 2, 100, 3, 4]
[1, 2, 100, 3, 4, [99, 98]]
[1, 2, 100, 3, 4, [99, 98], 'a', 'b', 'c']
```

Python Lists (10)

The screenshot shows the PyCharm IDE with a project named 'turtle'. The 'Project' view on the left lists files from 'ex_17' to 'turtle.'. The main editor displays two Python files, 'ex_21.py' and 'ex_22.py'. 'ex_21.py' contains code for list operations: del, remove, pop, and clear. 'ex_22.py' contains code for list operations: remove, pop, and clear. The 'Run' console at the bottom shows the output of the code, which is a list of lists: `[[2, 3, 4][3, 4]`, `[2, 3, 4, 3, 2, 1, 2][2, 3, 4, 3, 2, 1][2, 3, 4, 3, 1][`

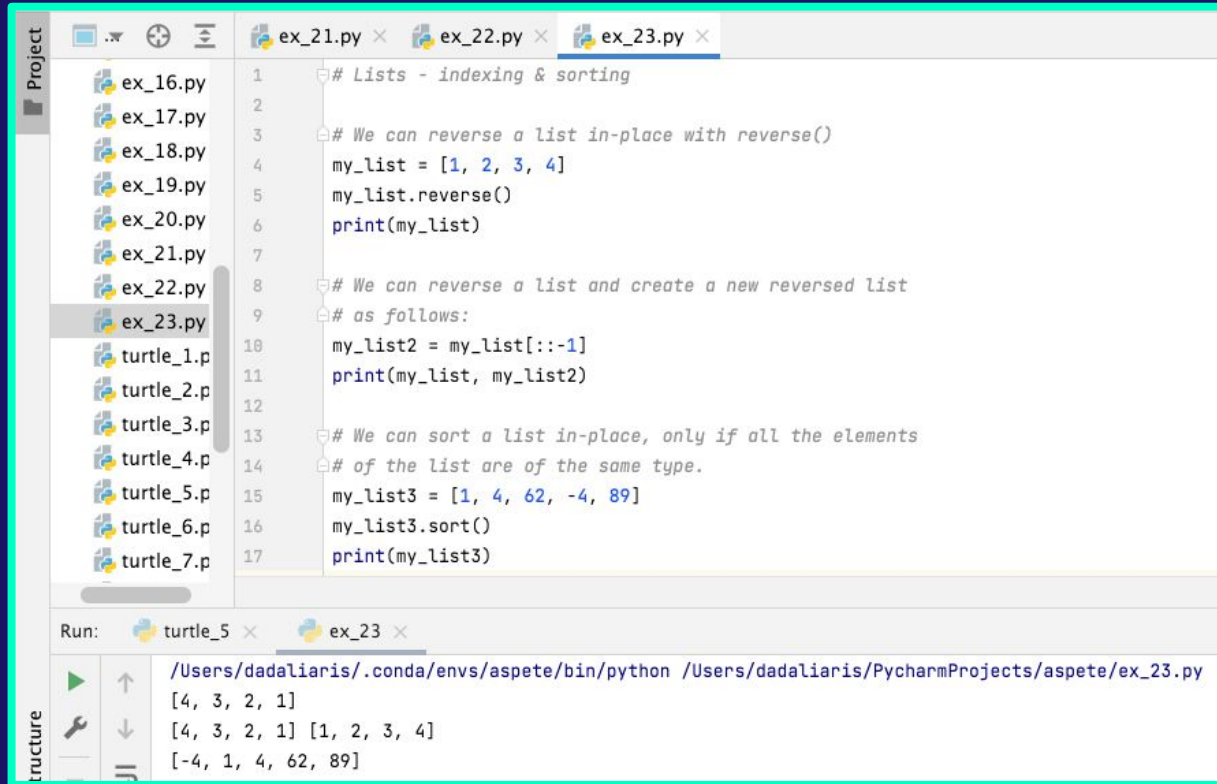
```
1 # Lists - del, remove, pop, clear ✓
2
3 my_list = [1, 2, 3, 4]
4 del my_list
5
6 my_list = [1, 2, 3, 4]
7 del my_list[:]
8 print(my_list, end="")
9
10 my_list = [1, 2, 3, 4]
11 del my_list[0]
12 print(my_list, end="")
13
14 my_list = [1, 2, 3, 4]
15 del my_list[0:2]
16 print(my_list)
17
```

```
18 my_list = [1, 2, 3, 4, 3, 2, 1, 2]
19 my_list.remove(1)
20 print(my_list, end="")
21
22 my_list.pop()
23 print(my_list, end="")
24
25 my_list.pop(4)
26 print(my_list, end="")
27
28 my_list.clear()
29 print(my_list)
30
```

Run: turtle_5 × ex_22 ×

```
/Users/dadalIaris/.conda/envs/aspete/bin/python /Users/dadalIaris/PycharmProjects/aspete/ex_22.py
[[2, 3, 4][3, 4]
[2, 3, 4, 3, 2, 1, 2][2, 3, 4, 3, 2, 1][2, 3, 4, 3, 1][
```

Python Lists (11)



The screenshot shows a PyCharm IDE window with three tabs: ex_21.py, ex_22.py, and ex_23.py. The main editor displays the code for ex_23.py, which demonstrates list indexing and sorting. The code is as follows:

```
1  # Lists - indexing & sorting
2
3  # We can reverse a list in-place with reverse()
4  my_list = [1, 2, 3, 4]
5  my_list.reverse()
6  print(my_list)
7
8  # We can reverse a list and create a new reversed list
9  # as follows:
10 my_list2 = my_list[::-1]
11 print(my_list, my_list2)
12
13 # We can sort a list in-place, only if all the elements
14 # of the list are of the same type.
15 my_list3 = [1, 4, 62, -4, 89]
16 my_list3.sort()
17 print(my_list3)
```

Below the code editor, the Run console shows the output for the execution of ex_23.py:

```
Run: turtle_5 x ex_23 x
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_23.py
[4, 3, 2, 1]
[4, 3, 2, 1] [1, 2, 3, 4]
[-4, 1, 4, 62, 89]
```

Python Lists (12)

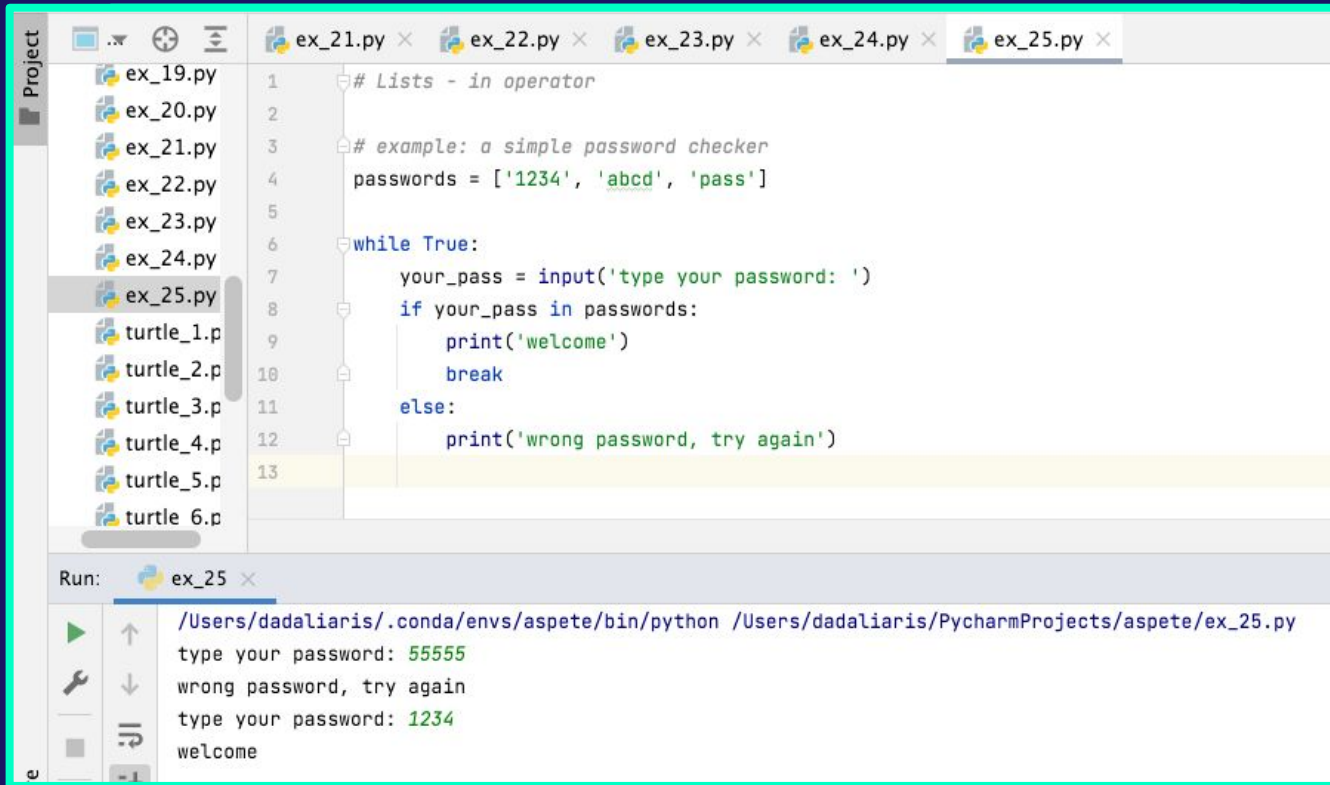
The screenshot shows a Python IDE with a project structure on the left and a code editor on the right. The code editor displays the following Python code:

```
1 # Lists - comparisons
2
3 list_a, list_b = [1, 2, 3, 4], [0, 2, 3, 4]
4 print(list_a > list_b)
5
6 list_a, list_b = [1, 2, 3, 'a'], [0, 2, 3, 'b']
7 print(list_a > list_b)
8 list_a, list_b = [1, 2, 3, 'a'], [1, 2, 3, 'b']
9 print(list_a > list_b)
10 list_a, list_b = [1, 2, 3, 5], [0, 2, 3, 'b']
11 print(list_a > list_b)
12
13 list_a, list_b = [1, 2, 3, 5], [1, 2, 3, 'b']
14 print(list_a > list_b)
```

The Run console at the bottom shows the execution of the code for `ex_24.py`. The output is:

```
Run: ex_24 ×
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_24.py
True
True
False
True
Traceback (most recent call last):
  File "/Users/dadaliaris/PycharmProjects/aspete/ex_24.py", line 16, in <module>
    print(list_a > list_b)
TypeError: '>' not supported between instances of 'int' and 'str'
```

Python Lists (13)



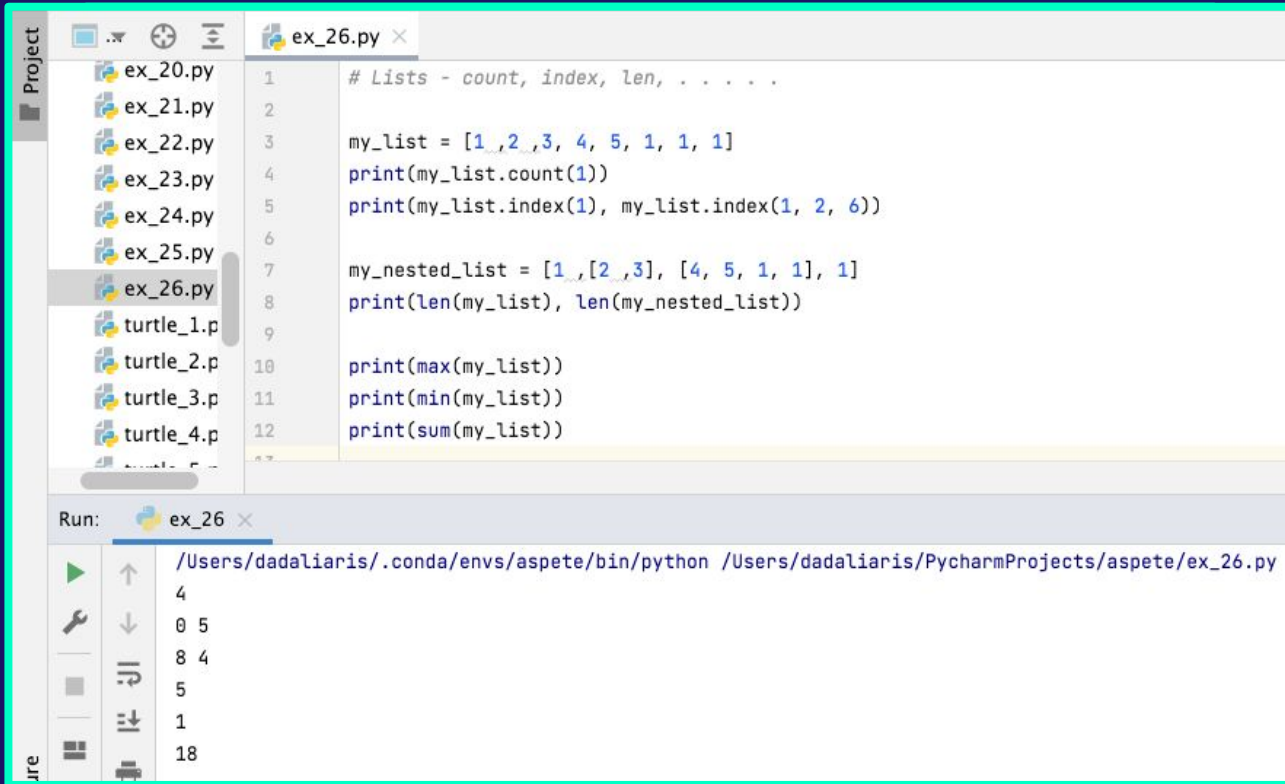
The screenshot shows a Python IDE with a project named 'Project' on the left. The main editor displays a Python script named 'ex_25.py' with the following code:

```
1  # Lists - in operator
2
3  # example: a simple password checker
4  passwords = ['1234', 'abcd', 'pass']
5
6  while True:
7      your_pass = input('type your password: ')
8      if your_pass in passwords:
9          print('welcome')
10         break
11     else:
12         print('wrong password, try again')
13
```

Below the editor, the 'Run' console shows the execution of 'ex_25.py' with the following output:

```
Run: ex_25 x
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_25.py
type your password: 55555
wrong password, try again
type your password: 1234
welcome
```


Python Lists (14)



The screenshot shows a Python IDE with a project view on the left and a code editor on the right. The project view lists several files, including 'ex_20.py' through 'ex_26.py' and 'turtle_1.p' through 'turtle_4.p'. The code editor displays the following Python code:

```
1 # Lists - count, index, len, . . . . .
2
3 my_list = [1, 2, 3, 4, 5, 1, 1, 1]
4 print(my_list.count(1))
5 print(my_list.index(1), my_list.index(1, 2, 6))
6
7 my_nested_list = [1, [2, 3], [4, 5, 1, 1], 1]
8 print(len(my_list), len(my_nested_list))
9
10 print(max(my_list))
11 print(min(my_list))
12 print(sum(my_list))
```

Below the code editor, the 'Run' console shows the execution of 'ex_26' with the following output:

```
Run: ex_26 x
/Users/dadalariis/.conda/envs/aspete/bin/python /Users/dadalariis/PycharmProjects/aspete/ex_26.py
4
0 5
8 4
5
1
18
```

Python Strings (1)

- A string is an ordered and immutable sequence of characters.
- There is no specific type for characters in Python. A character is simply a string with length equal to one.
- Strings can be declared using single quotes (' '), double quotes (" ") or triple single/double quotes (""" / """).
- Triple quotes can be used for multi-line strings and should only be used for commenting our code(documentation/docstring).

Python Strings (2)

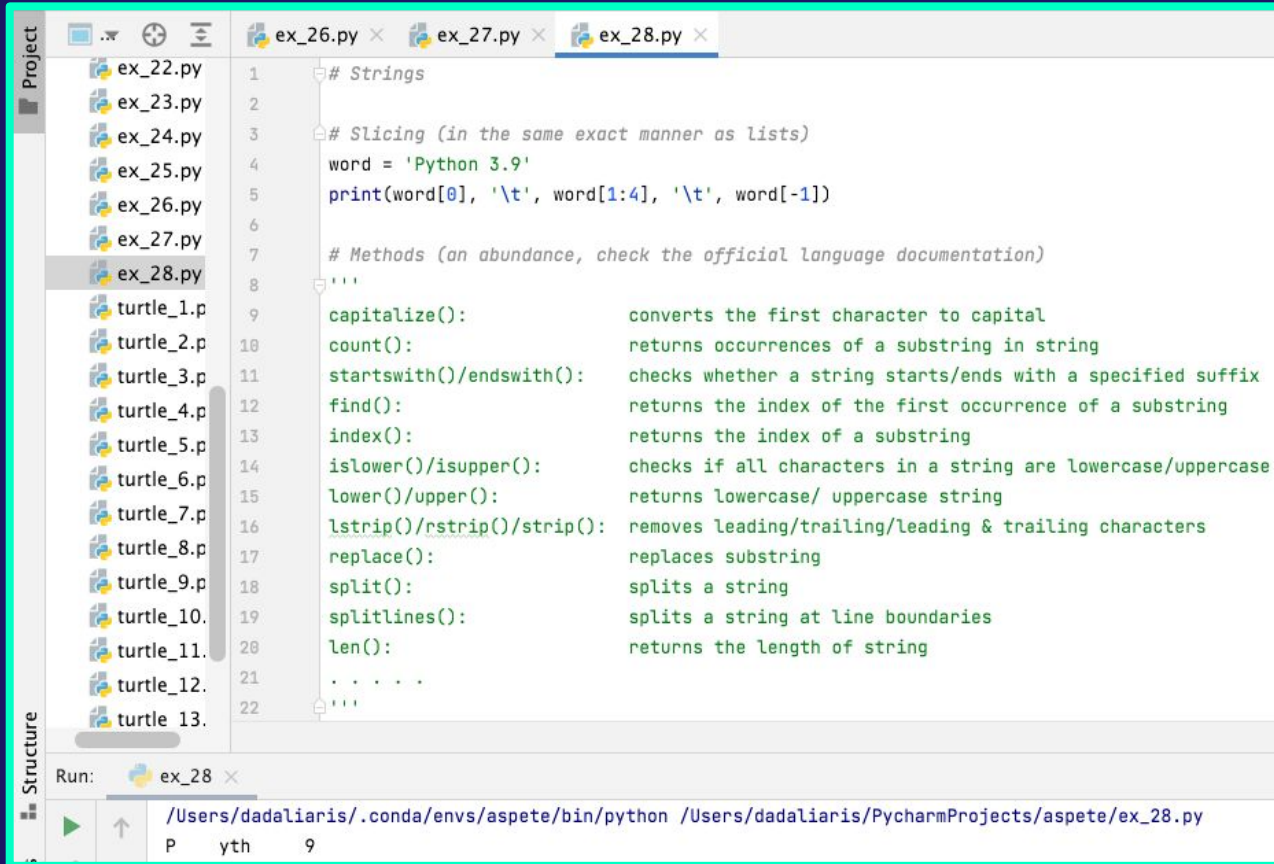
The image shows a code editor window with two tabs: `ex_26.py` and `ex_27.py`. The `ex_27.py` tab is active, displaying the following Python code:

```
1  # Strings
2
3  # Creating Strings
4  word = 'Python'
5  print(word, type(word))
6
7  word2 = str(123.45)
8  print(word2, type(word2))
9
10 # Built-in functions
11 print(max(word), "\t", min(word), "\t", sorted(word))
12
13 # Checking for the existence of characters
14 print('p' in word, '\t', 'P' in word, '\t', 'th' in word)
```

Below the code editor is a Run console window for `ex_27` showing the output of the code:

```
Run: ex_27 x
/Users/dadalariaris/.conda/envs/aspete/bin/python /Users/dadalariaris/PycharmProjects/aspete/ex_27.py
Python <class 'str'>
123.45 <class 'str'>
y P ['P', 'h', 'n', 'o', 't', 'y']
False True True
```

Python Strings (3)



The screenshot shows a code editor with three tabs: ex_26.py, ex_27.py, and ex_28.py. The active tab is ex_28.py, which contains the following code:

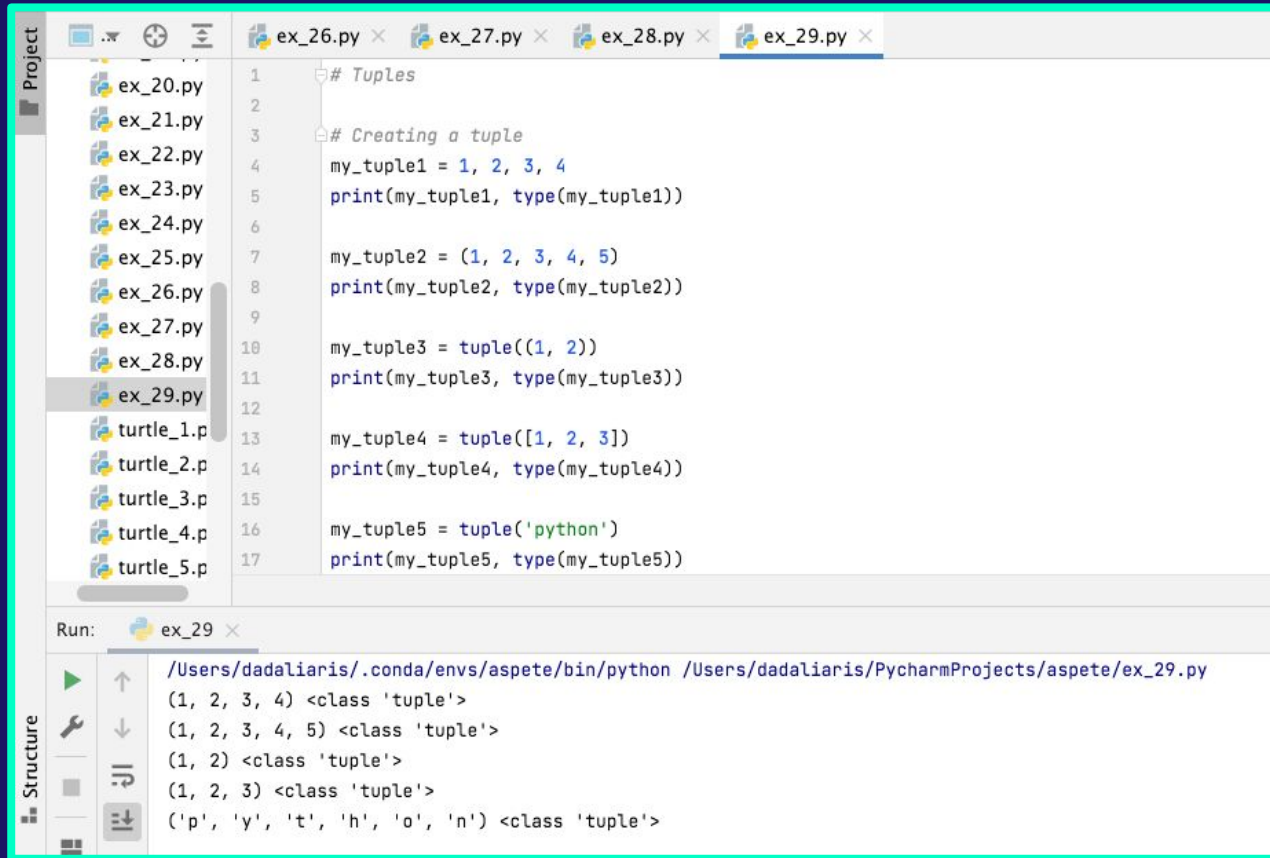
```
1  # Strings
2
3  # Slicing (in the same exact manner as lists)
4  word = 'Python 3.9'
5  print(word[0], '\t', word[1:4], '\t', word[-1])
6
7  # Methods (an abundance, check the official language documentation)
8  '''
9
10     capitalize():           converts the first character to capital
11     count():                returns occurrences of a substring in string
12     startswith()/endswith(): checks whether a string starts/ends with a specified suffix
13     find():                 returns the index of the first occurrence of a substring
14     index():                returns the index of a substring
15     islower()/isupper():    checks if all characters in a string are lowercase/uppercase
16     lower()/upper():        returns lowercase/ uppercase string
17     lstrip()/rstrip()/strip(): removes leading/trailing/leading & trailing characters
18     replace():              replaces substring
19     split():                splits a string
20     splitlines():           splits a string at line boundaries
21     len():                  returns the length of string
22     . . . . .
23     '''
```

The left sidebar shows a project structure with files ex_22.py through ex_28.py and turtle_1.p through turtle_13.p. The bottom status bar shows the command: `/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_28.py` with a Python icon and the number 9.

Python Tuples (1)

- A tuple is an immutable & iterable ordered sequence of element/objects.
- Immutable:
 - We can delete a tuple
 - We cannot delete/modify its content
- Similar methods to a list, except those that might change its content.

Python Tuples (2)



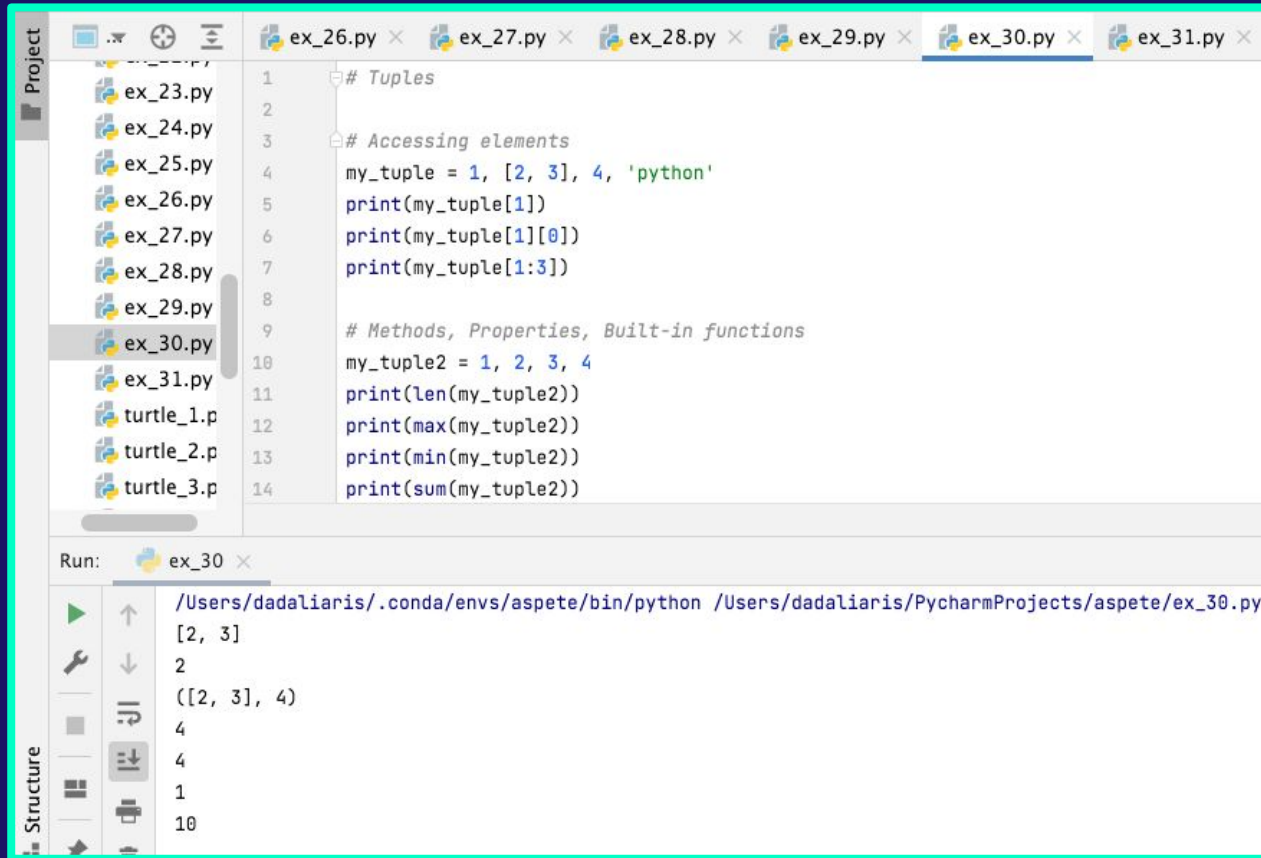
The screenshot shows the PyCharm IDE with a project named 'aspete'. The 'Project' view on the left lists files from ex_20.py to turtle_5.p. The 'Run' view at the bottom shows the execution of ex_29.py, displaying the output of the Python script.

```
1  # Tuples
2
3  # Creating a tuple
4  my_tuple1 = 1, 2, 3, 4
5  print(my_tuple1, type(my_tuple1))
6
7  my_tuple2 = (1, 2, 3, 4, 5)
8  print(my_tuple2, type(my_tuple2))
9
10 my_tuple3 = tuple((1, 2))
11 print(my_tuple3, type(my_tuple3))
12
13 my_tuple4 = tuple([1, 2, 3])
14 print(my_tuple4, type(my_tuple4))
15
16 my_tuple5 = tuple('python')
17 print(my_tuple5, type(my_tuple5))
```

Run: ex_29 ×

```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_29.py
(1, 2, 3, 4) <class 'tuple'>
(1, 2, 3, 4, 5) <class 'tuple'>
(1, 2) <class 'tuple'>
(1, 2, 3) <class 'tuple'>
('p', 'y', 't', 'h', 'o', 'n') <class 'tuple'>
```

Python Tuples (3)



The screenshot shows a Python IDE with a project explorer on the left containing files from ex_23.py to turtle_3.p. The main editor displays the code in ex_30.py, which demonstrates tuple creation and various operations. Below the editor, the Run console shows the execution of the code, displaying the output of each print statement.

```
1  # Tuples
2
3  # Accessing elements
4  my_tuple = 1, [2, 3], 4, 'python'
5  print(my_tuple[1])
6  print(my_tuple[1][0])
7  print(my_tuple[1:3])
8
9  # Methods, Properties, Built-in functions
10 my_tuple2 = 1, 2, 3, 4
11 print(len(my_tuple2))
12 print(max(my_tuple2))
13 print(min(my_tuple2))
14 print(sum(my_tuple2))
```

Run: ex_30 ×

```
/Users/dadaliaris/.conda/envs/aspete/bin/python /Users/dadaliaris/PycharmProjects/aspete/ex_30.py
[2, 3]
2
([2, 3], 4)
4
4
1
10
```

Python Dictionaries (1)

- A dictionary is mutable, iterable & unordered sequence of key-value pairs.
 - Keys must be unique
 - Key types must be immutable
 - Value can be of any type

Python Dictionaries (2)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Dictionaries """
2
3 my_dict1 = {}
4 print(my_dict1, type(my_dict1))
5
6 my_dict2 = dict()
7 print(my_dict2, type(my_dict2))
8
9 my_dict3 = {"a": "abcdef", "b": 2, 4: [1, 2, 4, 9]}
10 print(my_dict3, type(my_dict3))
11
12 my_dict4 = dict([("log100", 2), ("pi", 3.14)])
13 print(my_dict4, type(my_dict4))
14
15 letters = ["a", "b", "c", "d"]
16 numbers = [1, 2, 3, 4]
17 my_dict5 = dict(zip(letters, numbers))
18 print(my_dict5)
19 my_dict6 = dict(zip(numbers, letters))
20 print(my_dict6)
21
```

The IPython console on the right shows the output of the script:

```
wdir='C:/Users/antho/.spyder-py3')
{} <class 'dict'>
{} <class 'dict'>
{'a': 'abcdef', 'b': 2, 4: [1, 2, 4, 9]} <class 'dict'>
{'log100': 2, 'pi': 3.14} <class 'dict'>
{'a': 1, 'b': 2, 'c': 3, 'd': 4}

In [132]: runfile('C:/Users/antho/.spyder-py3/temp.py',
wdir='C:/Users/antho/.spyder-py3')
{} <class 'dict'>
{} <class 'dict'>
{'a': 'abcdef', 'b': 2, 4: [1, 2, 4, 9]} <class 'dict'>
{'log100': 2, 'pi': 3.14} <class 'dict'>
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
[1: 'a', 2: 'b', 3: 'c', 4: 'd']

In [133]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 21, Column: 1, Memory: 88 %.

Python Dictionaries (3)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 singer = {"Tool": "Keenan", "Pearl Jam": "Vedder", "QOTSA": "Homme"}
2
3 print(singer["QOTSA"])
4
5 for name in singer:
6     print(name, singer[name])
7
8 # When a search for a non-existed key is executed, a KeyError
9 # is generated in the console.
10 # print(singer["Clash"])
11
12 # Since Python 3.6 dictionaries are ordered, in the sense that
13 # the elements of a dictionary are presented in the way they
14 # were added.
15
16
```

The Python console on the right shows the output of the script:

```
In [4]: runfile('C:/Users/antho/.spyder-py3/temp.py',
wdir='C:/Users/antho/.spyder-py3')
Homme
Tool Keenan
Pearl Jam Vedder
QOTSA Homme

In [5]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 16, Column: 1, Memory: 84 %.

Python Dictionaries (4)

The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script named `temp.py` with the following code:

```
1 """ Dictionary Methods: keys / values /items """
2
3 singers = {"Tool":"Keenan", "Pearl Jam":"Vedder", "QOTSA":"Homme"}
4
5 for singer in singers.values():
6     print(singer)
7 print("\n")
8
9 for band in singers.keys():
10    print(band)
11 print("\n")
12
13 for band, singer in singers.items():
14    print("band name:", band, "\tsinger name:", singer)
15
```

The Python console on the right shows the output of the script:

```
In [16]: runfile('C:/Users/antho/.spyder-py3/temp.py',
wdir='C:/Users/antho/.spyder-py3')
Keenan
Vedder
Homme

Tool
Pearl Jam
QOTSA

band name: Tool           singer name: Keenan
band name: Pearl Jam     singer name: Vedder
band name: QOTSA         singer name: Homme

In [17]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 15, Column: 5, Memory: 87 %.

Python Dictionaries (5)

The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script named `temp.py` with the following code:

```
1 """ Dictionaries: data retrieval """
2
3 singers = {"Tool": "Keenan", "Pearl Jam": "Vedder", "QOTSA": "Homme"}
4
5 # Classic way:
6 print(singers["Tool"])
7 # print(singers["Clash"])
8
9 # Using the get() method:
10 print(singers.get("Tool"))
11 print(singers.get("Clash"))
12 print(singers.get("Clash", "no such key"))
13
14 # Checking for a key (using the membership operator):
15 print("Tool" in singers)
16 print("Clash" in singers)
17
```

The Python console on the right shows the output of the script:

```
In [18]: runfile('C:/Users/antho/.spyder-py3/temp.py',
wdir='C:/Users/antho/.spyder-py3')
Keenan
Keenan
None
no such key

In [19]: runfile('C:/Users/antho/.spyder-py3/temp.py',
wdir='C:/Users/antho/.spyder-py3')
Keenan
Keenan
None
no such key
True
False

In [20]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 17, Column: 1, Memory: 80 %.

Python Dictionaries (6)

The screenshot shows the Spyder Python IDE interface. The editor window displays a Python script named `temp.py` with the following code:

```
1 """ Dictionaries: data entry & modification """
2
3 singers = {"Tool": "Keenan", "Pearl Jam": "Vedder", "QOTSA": "Homme"}
4
5 singers["Radiohead"] = "Yorke"
6 print(singers)
7 print("\n")
8
9 singers["Pearl Jam"] = ["Vedder", "Buckley"]
10 print(singers)
11 print(singers["Pearl Jam"])
12 print(singers["Pearl Jam"][0])
13
14
```

The Python console on the right shows the output of the script:

```
'Homme', 'Radiohead': 'Yorke'}
{'Tool': 'Keenan', 'Pearl Jam': ['Vedder', 'Buckley'],
'QOTSA': 'Homme', 'Radiohead': 'Yorke'}
['Vedder', 'Buckley']

In [24]: runfile('C:/Users/antho/.spyder-py3/temp.py',
wdir='C:/Users/antho/.spyder-py3')
{'Tool': 'Keenan', 'Pearl Jam': 'Vedder', 'QOTSA':
'Homme', 'Radiohead': 'Yorke'}

{'Tool': 'Keenan', 'Pearl Jam': ['Vedder', 'Buckley'],
'QOTSA': 'Homme', 'Radiohead': 'Yorke'}
['Vedder', 'Buckley']
Vedder

In [25]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 14, Column: 1, Memory: 83 %.

Python Dictionaries (7)

The screenshot shows the Spyder Python IDE interface. The editor window displays a Python script named `temp.py` with the following code:

```
1 """ Dictionaries: copying dictionaries """
2
3 singers = {"Tool": "Keenan", "Pearl Jam": "Vedder", "QOTSA": "Homme"}
4 singers2 = singers
5
6 singers2["QOTSA"] = "Buckley"
7 print(singers)
8 print(singers2)
9 print("\n")
10
11 from copy import deepcopy
12 singers3 = deepcopy(singers)
13 singers3["QOTSA"] = "J.Homme"
14 print(singers)
15 print(singers3)
16
17
```

The Python console on the right shows the output of the script:

```
In [27]: runfile('C:/Users/antho/.spyder-py3/temp.py',
wdir='C:/Users/antho/.spyder-py3')
{'Tool': 'Keenan', 'Pearl Jam': 'Vedder', 'QOTSA':
'Buckley'}
{'Tool': 'Keenan', 'Pearl Jam': 'Vedder', 'QOTSA':
'Buckley'}

{'Tool': 'Keenan', 'Pearl Jam': 'Vedder', 'QOTSA':
'Buckley'}
{'Tool': 'Keenan', 'Pearl Jam': 'Vedder', 'QOTSA':
'J.Homme'}

In [28]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 16, Column: 1, Memory: 85 %.

Python Dictionaries (8)

The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script named `temp.py` with the following code:

```
1 """ Dictionaries: deleting data (del, pop, popitem, clear) """
2
3 singers = {"Tool": "Keenan",
4           "Pearl Jam": "Vedder",
5           "QOTSA": "Homme",
6           "Soundgarden": "Cornell",
7           "Black Flag": "Rollins"}
8
9 del(singers["Tool"])
10 print(singers, "\n")
11
12 black_flag_singer=singers.pop("Black Flag")
13 print(singers)
14 print(black_flag_singer, "\n")
15
16 band_and_singer=singers.popitem()
17 print(singers)
18 print(band_and_singer, "\n")
19
20 singers.clear()
21 print(singers)
22
23
```

The right-hand side of the IDE contains several panels. At the top is a 'Usage' panel with the text: 'Here you can get help of any object'. Below it are tabs for 'Variable explorer', 'File explorer', and 'Help'. The 'Python console' panel shows the output of the script's execution:

```
In [34]: runfile('C:/Users/antho/.spyder-py3/temp.py',
wdir='C:/Users/antho/.spyder-py3')
{'Pearl Jam': 'Vedder', 'QOTSA': 'Homme', 'Soundgarden':
'Cornell', 'Black Flag': 'Rollins'}

{'Pearl Jam': 'Vedder', 'QOTSA': 'Homme', 'Soundgarden':
'Cornell'}
Rollins

{'Pearl Jam': 'Vedder', 'QOTSA': 'Homme'}
('Soundgarden', 'Cornell')

{}

In [35]:
```

At the bottom of the IDE, a status bar displays: 'Permissions: RW', 'End-of-lines: CRLF', 'Encoding: ASCII', 'Line: 23', 'Column: 1', and 'Memory: 81%'.

Python Dictionaries (9)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Dictionaries: sorting data """
2
3 albums = {"Tool":4, "Pearl Jam":8, "QOTSA":7, "Soundgarden":6, "Black Flag":3}
4
5 bands_sorted = sorted(albums.keys())
6 print(bands_sorted)
7
8 bands_sorted_r = sorted(albums.keys(), reverse=True)
9 print(bands_sorted_r)
10
11 print("\n")
12
13 albums_sorted = sorted(albums.values())
14 print(albums_sorted)
15
16 albums_sorted_r = sorted(albums.values(), reverse=True)
17 print(albums_sorted_r)
18
```

The Python console on the right shows the execution results for two input prompts:

```
In [39]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
['Black Flag', 'Pearl Jam', 'QOTSA',
'Soundgarden', 'Tool']
['Tool', 'Soundgarden', 'QOTSA', 'Pearl Jam',
'Black Flag']

[3, 4, 6, 7, 8]
[8, 7, 6, 4, 3]

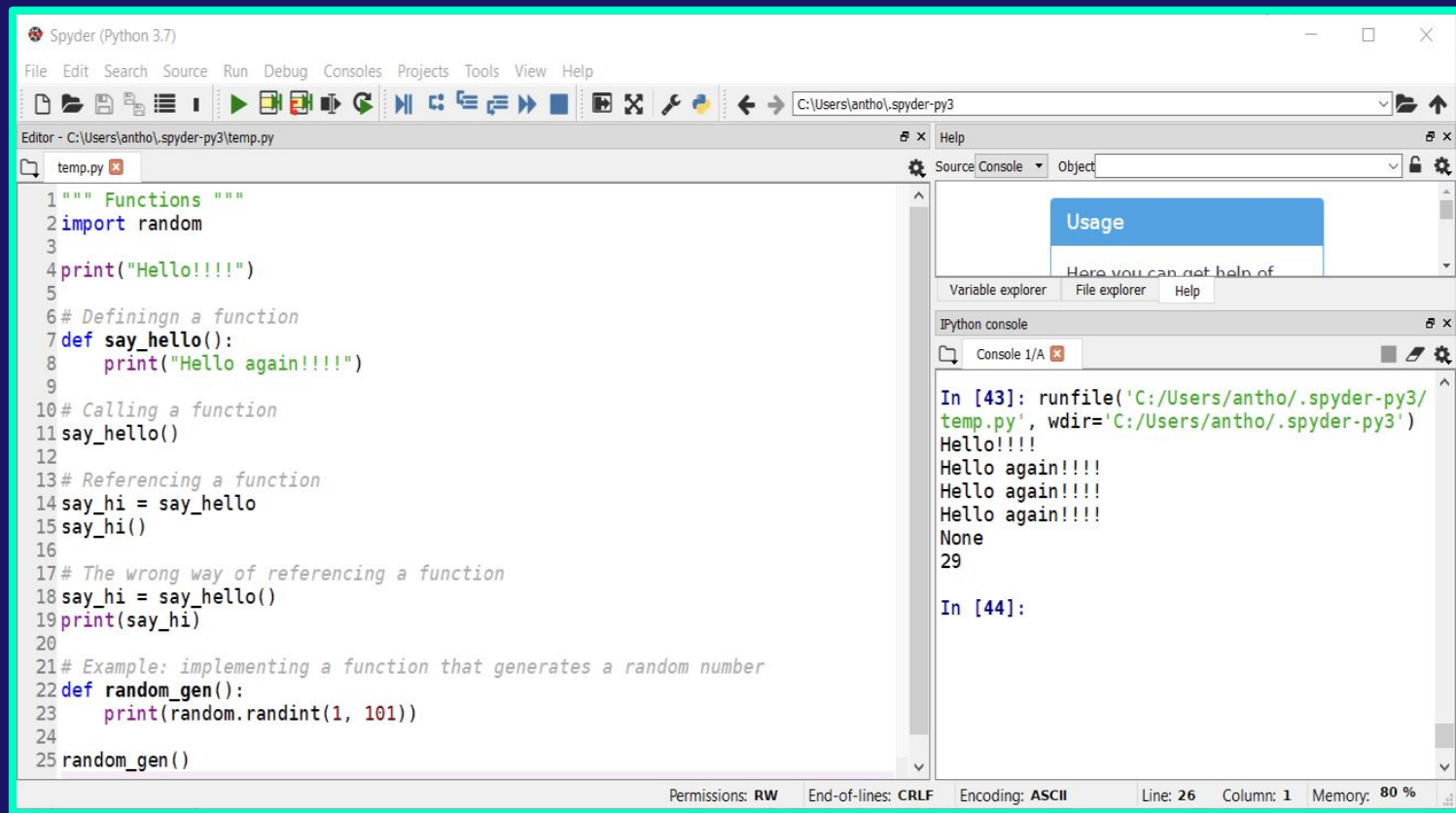
In [40]:
```

The status bar at the bottom of the IDE indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 18, Column: 1, Memory: 76 %.

Python Functions (1)

- Functional Programming:
 - Hierarchical design
 - Smaller problems, easier to be solved
 - Development, debugging & maintaining effort & time
 - Less code (reusability)
 - Readability
 - Abstraction
- `def function_name(parameters):`
 `commands`
 `.....`
 `commands`
 `Return_statement`
- parameters & return_statement are optional

Python Functions (2)



The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script named `temp.py` with the following code:

```
1 """ Functions """
2 import random
3
4 print("Hello!!!!")
5
6 # Defining a function
7 def say_hello():
8     print("Hello again!!!!")
9
10 # Calling a function
11 say_hello()
12
13 # Referencing a function
14 say_hi = say_hello
15 say_hi()
16
17 # The wrong way of referencing a function
18 say_hi = say_hello()
19 print(say_hi)
20
21 # Example: implementing a function that generates a random number
22 def random_gen():
23     print(Random.randint(1, 101))
24
25 random_gen()
```

The IPython console on the right shows the execution output for the script:

```
In [43]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
Hello!!!!
Hello again!!!!
Hello again!!!!
Hello again!!!!
None
29

In [44]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 26, Column: 1, Memory: 80 %.

Python Functions (3)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Scope """
2
3 # Namespaces: module namespace, local namespace, built-in namespace
4
5 # Example: module namespace
6 import random
7
8 def random_gen():
9     print(random.randint(1, 101))
10
11 random_gen()
12 print(random.randint(1, 101))
13
14 # Example: local namespace
15 a = 100
16
17 def print_num():
18     a = 200
19     print("a inside this function is equal to",a)
20
21 print_num()
22 print("a outside this function is equal to",a)
23
```

The right-hand side of the IDE contains several panels. At the top is a 'Usage' panel with the text 'Here you can get help of'. Below it are 'Variable explorer', 'File explorer', and 'Help' tabs. The 'IPython console' panel shows the execution of the script:

```
In [47]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
36
84
a inside this function is equal to 200
a outside this function is equal to 100

In [48]:
```

The status bar at the bottom of the IDE indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 23, Column: 1, Memory: 80%.

Python Functions (4)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Scope """
2
3 # Example: global keyword
4
5 # This is a global variable
6 my_var = 0
7
8 def add_two():
9     # Referencing the variable outside this function
10    global my_var
11    my_var = my_var + 2
12    print("inside add_two():", my_var)
13
14 add_two()
15 print("outside the function:", my_var)
16
```

The IPython console on the right shows the execution results:

```
In [50]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
inside add_two(): 2
outside the function: 2

In [51]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 16, Column: 1, Memory: 87 %.

Python Functions (5)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Internal & Embedded Functions """
2
3 # Example #1:
4 def func_1():
5     print("func_1")
6
7 def func_2():
8     print("func_2")
9     func_1()
10
11 func_2()
12 func_1()
13 print("\n")
14
15 # Example #2:
16 def func_3():
17     print("external function")
18     def func_4():
19         print("internal function")
20     func_4()
21
22 func_3()
23
```

The IPython console on the right shows the execution of the script:

```
In [52]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
func_2
func_1
func_1

external function
internal function

In [53]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 23, Column: 1, Memory: 80 %.

Python Functions (6)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Internal & Embedded Functions """
2
3 # Example #3:
4 x = 100
5
6 def func_5():
7     x = 200
8     print(x)
9
10    def func_6():
11        x = 300
12        print(x)
13
14    func_6()
15    print(x)
16
17 func_5()
18 print(x)
19
```

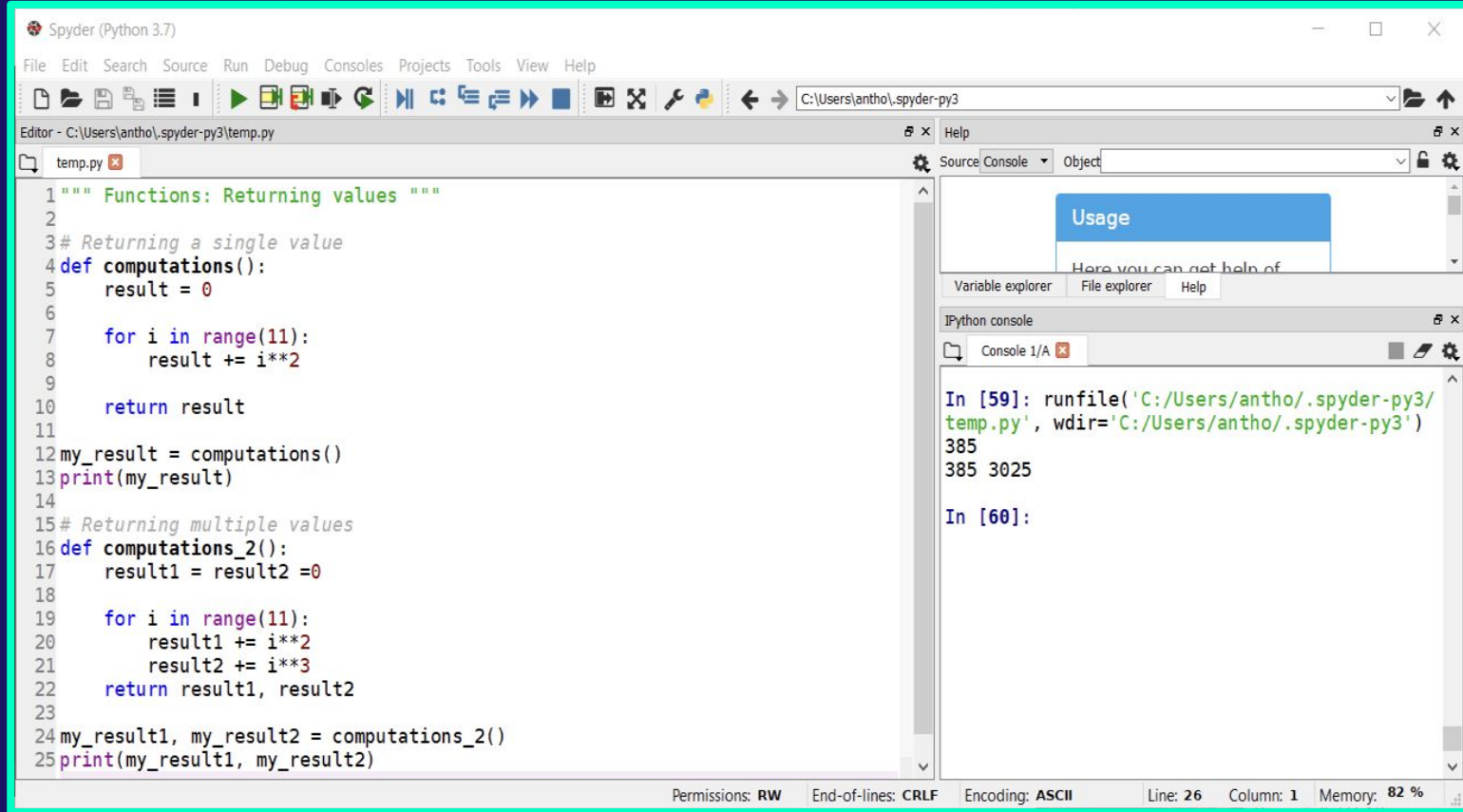
The IPython console on the right shows the execution of the script:

```
In [53]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
200
300
200
100

In [54]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 19, Column: 1, Memory: 80%.

Python Functions (7)



The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Returning values """
2
3 # Returning a single value
4 def computations():
5     result = 0
6
7     for i in range(11):
8         result += i**2
9
10    return result
11
12 my_result = computations()
13 print(my_result)
14
15 # Returning multiple values
16 def computations_2():
17     result1 = result2 = 0
18
19     for i in range(11):
20         result1 += i**2
21         result2 += i**3
22     return result1, result2
23
24 my_result1, my_result2 = computations_2()
25 print(my_result1, my_result2)
```

The right-hand side of the IDE contains several panels. At the top, there is a "Usage" panel with the text "Here you can get help of". Below it are the "Variable explorer", "File explorer", and "Help" panels. The "IPython console" panel is active, showing the execution of the script:

```
In [59]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
385
385 3025

In [60]:
```

The status bar at the bottom of the IDE indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 26, Column: 1, Memory: 82 %.

Python Functions (8)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Parameters """
2
3 # Create a function that returns only the negative numbers of
4 # a sequence of numbers.
5 def negatives():
6     neg_list = list()
7
8     for item in my_sequence:
9         if item < 0:
10             neg_list.append(item)
11
12     return neg_list
13
14 my_sequence = [-1, -2, 3, 4, -5]
15 res1 = negatives()
16 print(res1)
17
18 my_sequence = (-1, -2, 3, 4, -5)
19 res2 = negatives()
20 print(res2)
21
22 my_sequence = {-1, -2, 3, 4, -5}
23 res3 = negatives()
24 print(res3)
25
```

The right-hand side of the IDE contains several panels. The top panel is titled "Usage" and contains the text "Here you can get help of". Below it are tabs for "Variable explorer", "File explorer", and "Help". The bottom panel is the "Python console", which shows the execution of the script:

```
In [63]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
[-1, -2, -5]
[-1, -2, -5]
[-2, -5, -1]

In [64]:
```

The status bar at the bottom of the IDE indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 25, Column: 1, Memory: 81 %.

Python Functions (9)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Parameters """
2
3 # Create a function that returns only the negative numbers of
4 # a sequence of numbers.
5 def negatives(some_sequence):
6     neg_list = list()
7
8     for item in some_sequence:
9         if item < 0:
10            neg_list.append(item)
11
12    return neg_list
13
14 my_sequence = [-1, -2, 3, 4, -5]
15 res1 = negatives(my_sequence)
16 print(res1)
17
18 my_sequence = (-1, -2, 3, 4, -5)
19 res2 = negatives(my_sequence)
20 print(res2)
21
22 my_sequence = {-1, -2, 3, 4, -5}
23 res3 = negatives(my_sequence)
24 print(res3)
25
```

The right-hand side of the IDE contains several panels. At the top is a "Usage" panel with the text "Here you can get help of". Below it are tabs for "Variable explorer", "File explorer", and "Help". The "Python console" panel is active, showing the following execution results:

```
In [63]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
[-1, -2, -5]
[-1, -2, -5]
[-2, -5, -1]

In [64]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
[-1, -2, -5]
[-1, -2, -5]
[-2, -5, -1]

In [65]:
```

The status bar at the bottom of the IDE shows: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 25, Column: 1, Memory: 80 %.

Python Functions (10)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Parameters """
2
3 # Create a function that takes a list of numbers and returns
4 # the maximum, minimum and average value.
5 def max_min_avg(my_list):
6     min_num = min(my_list)
7     max_num = max(my_list)
8     avg_num = sum(my_list)/len(my_list)
9
10    return min_num, max_num, avg_num
11
12 some_list = [-2, -1, 0, 1, 2]
13
14 var_a, var_b, var_c = max_min_avg(some_list)
15 print(var_a, var_b, var_c)
16
17 all_results = max_min_avg(some_list)
18 print(all_results, type(all_results))
19
```

The right-hand side of the IDE contains several panels. At the top is a 'Usage' panel with a blue header and the text 'Here you can get help of'. Below it are tabs for 'Variable explorer', 'File explorer', and 'Help'. The 'Python console' panel is active, showing the following output:

```
In [65]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
-2 2 0.0
(-2, 2, 0.0) <class 'tuple'>

In [66]:
```

The status bar at the bottom of the IDE indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 19, Column: 1, Memory: 74%.

Python Functions (11)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Parameters """
2
3 # Values are passed by object in Python
4 def func(par_a, par_b, par_c):
5     print("inside this function")
6     par_c.append(1)
7     par_a = []
8     par_b.append(1)
9     par_c = []
10    print("par_a, par_b, par_c", par_a, par_b, par_c)
11
12 a, b, c = [1, 2], [3, 4], [5, 6]
13 print("before function call")
14 print(a, b, c)
15 print("\n")
16
17 func(a, b, c)
18 print("after function call")
19 print(a, b, c)
20
```

The Python console on the right shows the execution output:

```
In [68]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
before function call
[1, 2] [3, 4] [5, 6]

inside this function
par_a, par_b, par_c [] [3, 4, 1] []
after function call
[1, 2] [3, 4, 1] [5, 6, 1]

In [69]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 20, Column: 1, Memory: 79 %.

Python Functions (12)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Parameters & Default Values """
2
3 # Beware: default arguments should always follow non-default arguments
4 def greetings(name, greet="Hello", question="how are you?"):
5     print(greet, name, question)
6
7 greetings("Andy")
8 greetings("Andy", "Hey")
9 greetings("Andy", "Hey", "how are you today?")
10 print("\n")
11
12 greetings(greet="Hello", question="how are you?", name="Andy")
13 greetings(question="how are you?", name="Andy", greet="Hey")
14
15 # The following commands will fail . . . .
16 # greetings()
17 # greetings(greet="Hey", question="how are you today?")
18
```

The IPython console on the right shows the output of running the script:

```
In [87]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
Hello Andy how are you?
Hey Andy how are you?
Hey Andy how are you today?

Hello Andy how are you?
Hey Andy how are you?

In [88]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 18, Column: 1, Memory: 80 %.

Python Functions (13)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
1 """ Functions: Parameters Packing """
2
3 def avg(*numbers):
4     sum_of_numbers = 0
5
6     for num in numbers:
7         sum_of_numbers += num
8
9     return sum_of_numbers/len(numbers)
10
11 print(avg(1, 2, 3, 4))
12 print(avg(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
13
14 def grades(name, **courses):
15     print(name, ":")
16
17     for course, grade in courses.items():
18         print(course, "=", "grade")
19
20
21 grades("Andy", Python=7)
22 grades("Jeff", Python=9, Physics=10)
23
```

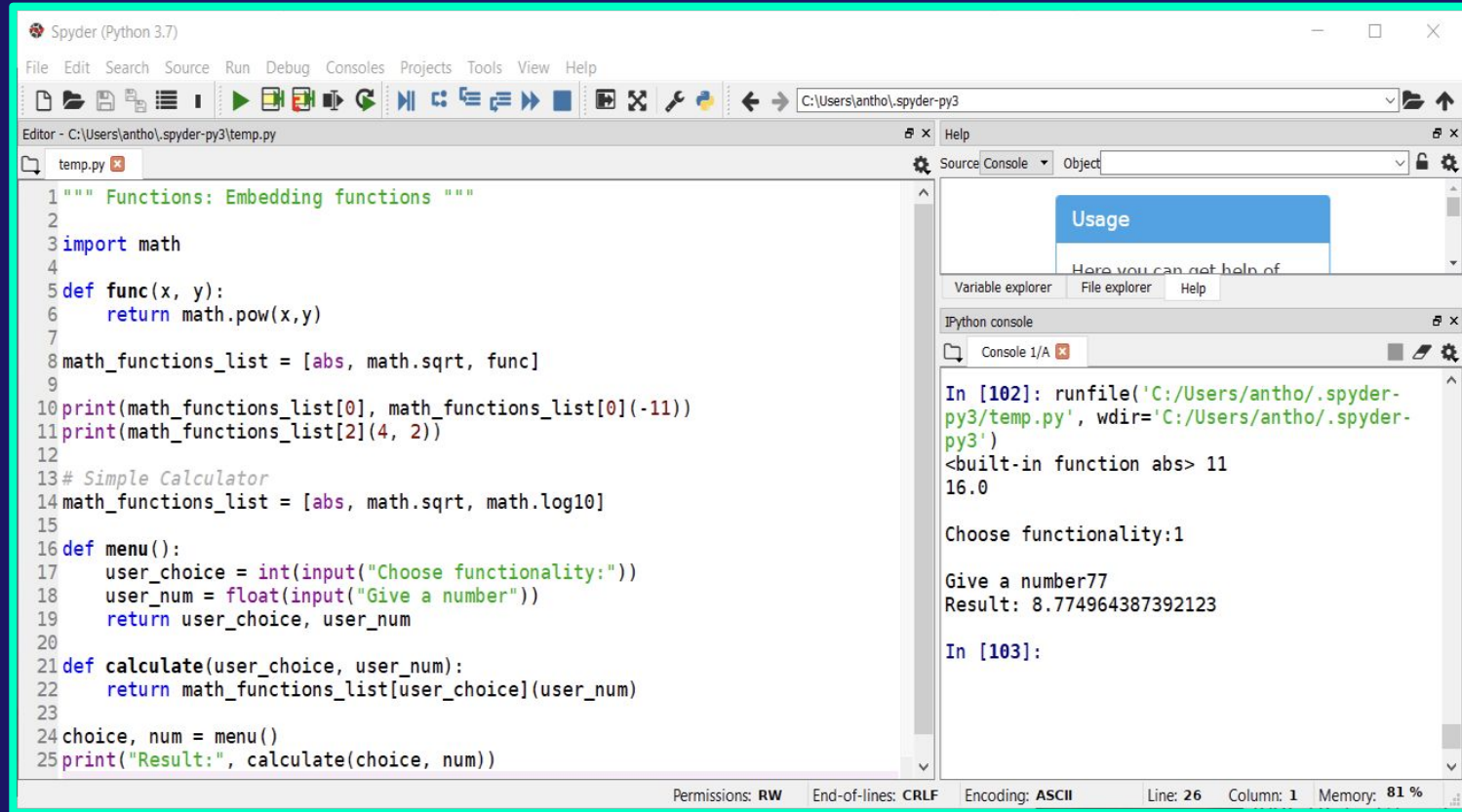
The IPython console on the right shows the execution of the script:

```
In [96]: runfile('C:/Users/antho/.spyder-py3/
temp.py', wdir='C:/Users/antho/.spyder-py3')
2.5
5.5
Andy :
Python = grade
Jeff :
Python = grade
Physics = grade

In [97]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 23, Column: 1, Memory: 75 %.

Python Functions (14)



The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script named `temp.py` with the following code:

```
1 """ Functions: Embedding functions """
2
3 import math
4
5 def func(x, y):
6     return math.pow(x,y)
7
8 math_functions_list = [abs, math.sqrt, func]
9
10 print(math_functions_list[0], math_functions_list[0](-11))
11 print(math_functions_list[2](4, 2))
12
13 # Simple Calculator
14 math_functions_list = [abs, math.sqrt, math.log10]
15
16 def menu():
17     user_choice = int(input("Choose functionality:"))
18     user_num = float(input("Give a number"))
19     return user_choice, user_num
20
21 def calculate(user_choice, user_num):
22     return math_functions_list[user_choice](user_num)
23
24 choice, num = menu()
25 print("Result:", calculate(choice, num))
```

The Python console on the right shows the execution output:

```
In [102]: runfile('C:/Users/antho/.spyder-
py3/temp.py', wdir='C:/Users/antho/.spyder-
py3')
<built-in function abs> 11
16.0

Choose functionality:1

Give a number77
Result: 8.774964387392123

In [103]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 26, Column: 1, Memory: 81 %.

Python Functions (15)

The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script named `temp.py` with the following code:

```
1 """ Functions: lambda functions, a special case of functions """
2
3 """
4 - We always start with the keyword lambda
5 - We can use parameters
6 - We can include only a single command
7 - They can be embedded
8 - They are defined and executed in-line
9 """
10
11 my_list = [-2, -1, 0, 1, 2, 4]
12
13 cubed_numbers = lambda x:x**3
14 cubed_list = [cubed_numbers(x) for x in my_list]
15 print(cubed_list)
16
17 positive_numbers = lambda x: x>0
18 print(list(filter(positive_numbers, my_list)))
19
20
```

The right-hand side of the IDE contains several panels. The `Help` panel is active, showing a `Usage` section with the text "Here you can get help of". Below it are the `Variable explorer`, `File explorer`, and `Help` tabs. The `IPython console` panel shows the execution of the script:

```
In [108]: runfile('C:/Users/antho/.spyder-py3/temp.py', wdir='C:/Users/antho/.spyder-py3')
[-8, -1, 0, 1, 8, 64]
[1, 2, 4]

In [109]:
```

The status bar at the bottom of the IDE displays the following information: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 20, Column: 1, Memory: 85%.

Python Modules (1)

- Python implements the concept of modular programming through modules.
- Module:
 - A Python file (or object file) in our computer that contains code that is available for usage.
 - User-defined modules
 - Built-in modules
 - Code repositories (pip, conda,)

Python Modules (2)

The screenshot shows the Spyder Python IDE interface. The main editor displays two Python files: `area_computation.py` and `temp.py`.

area_computation.py (left pane):

```
1 """
2 A module for computing the
3 area of various shapes.
4 """
5
6 pi = 3.14
7
8 def square_area(a=1):
9     return a*a
10
11 def rect_area(a=1, b=1):
12     return a*b
13
14
```

temp.py (right pane):

```
1 import area_computation
2
3 print(area_computation.pi)
4 print(area_computation.rect_area(10, 11))
5
6
7 # Alternatively you can import a module using an alias:
8 # import area_computation as ac
9
10 # or import specific functions:
11 # from area_computation import pi
12 # square_area(5)
13
14 # Packages are practically folders containing modules.
15 # Let's say we have a folder called foo that contains
16 # two modules called mod1 and mod2. The proper way
17 # to import these modules would be:
18 # import foo.mod1, foo.mod2
19
```

Variable explorer (top right):

Name	Type	Size	Value
i	str	1	pausing for the 2nd time
iter_var_etc		1	pausing for the

IPython console (bottom right):

```
In [69]: runfile('C:/Users/antho/.spyder-py3/temp.py',
wdir='C:/Users/antho/.spyder-py3')
Reloaded modules:
area_computation
3.14
110

In [70]:
```

Status bar (bottom):

Permissions: RW | End-of-lines: CRLF | Encoding: ASCII | Line: 19 | Column: 1 | Memory: 91 %

Python File Handling (1)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `untitled0.py` with the following code:

```
1 # File Handling
2
3 # Creating a file object in order to manipulate
4 # the contents of the file.
5 file_object = open("ibm02.nodes")
6
7 # Getting all the data stored in the file.
8 file_data = file_object.read()
9
10 print(file_data)
11
12 # Closing the file / Releasing the file object
13 # Files opened with the open() must always
14 # close, in order to avoid data corruption
15 # and release the allocated memory.
16 file_object.close()
17
```

The output window shows the result of the script execution, which is a list of 30 alphanumeric strings:

```
1 a0 14 16
2 a1 4 16
3 a2 14 16
4 a3 8 16
5 a4 6 16
6 a5 4 16
7 a6 6 16
8 a7 14 16
9 a8 4 16
10 a9 14 16
11 a10 6 16
12 a11 6 16
13 a12 4 16
14 a13 6 16
15 a14 16 16
16 a15 8 16
17 a16 14 16
18 a17 6 16
19 a18 6 16
20 a19 14 16
21 a20 4 16
22 a21 6 16
23 a22 14 16
24 a23 10 16
25 a24 8 16
26 a25 6 16
27 a26 6 16
28 a27 6 16
29 a28 16 16
30 a29 14 16
31 a30 6 16
```

The IPython console window shows the output of the `print` statement, displaying the same list of alphanumeric strings:

```
a19174 14 16
a19175 4 16
a19176 14 16
a19177 12 16
a19178 6 16
a19179 6 16
a19180 14 16
a19181 2 16
a19182 6 16
a19183 6 16
a19184 8 16
a19185 14 16
a19186 4 16
a19187 6 16
a19188 6 16
a19189 8 16
a19190 4 16
a19191 14 16
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 17, Column: 1, Memory: 83 %.

Python File Handling (2)

The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script for file handling. The code includes comments and operations to open a file, read its contents, and close it. The variable explorer shows a list of characters from 'a0' to 'a31'. The IPython console shows the output of running the script, displaying the characters 'a0' and '14'.

```
1 # File Handling
2
3 # Creating a file object in order to manipulate
4 # the contents of the file.
5 file_object = open("ibm02.nodes")
6
7 # Getting all the data stored in the file.
8 file_data = file_object.read(10)
9 print(file_data)
10
11 more_file_data = file_object.read(12)
12 print(more_file_data)
13
14 # Closing the file / Releasing the file object
15 # Files opened with the open() must always
16 # close, in order to avoid data corruption
17 # and release the allocated memory.
18 file_object.close()
19
```

1	a0	14	16
2	a1	4	16
3	a2	14	16
4	a3	8	16
5	a4	6	16
6	a5	4	16
7	a6	6	16
8	a7	14	16
9	a8	4	16
10	a9	14	16
11	a10	6	16
12	a11	6	16
13	a12	4	16
14	a13	6	16
15	a14	16	16
16	a15	8	16
17	a16	14	16
18	a17	6	16
19	a18	6	16
20	a19	14	16
21	a20	4	16
22	a21	6	16
23	a22	14	16
24	a23	10	16
25	a24	8	16
26	a25	6	16
27	a26	6	16
28	a27	6	16
29	a28	16	16
30	a29	14	16
31	a30	6	16

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the

Variable explorer File explorer Help

IPython console

Console 1/A

```
In [25]: runfile('C:/Users/antho/Desktop/backup_files/dev/untitled0.py', wdir='C:/Users/antho/Desktop/backup_files/dev')
a0
14

In [26]:
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 19 Column: 1 Memory: 87 %

Python File Handling (3)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `untitled0.py` with the following code:

```
1 # File Handling
2
3 file_object = open("ibm02.nodes")
4
5 for i in range(5):
6     line = file_object.readline()
7     print(line)
8
9 file_object.close()
10
11 # Line-by-line reading
12 file_object = open("ibm02.nodes")
13
14 line = file_object.readline()
15 print(line)
16
17 while line:
18     line = file_object.readline()
19     print(line)
20
21 file_object.close()
22
```

The variable explorer on the right shows a list of variables from `a0` to `a30`, each with a value and a memory address (e.g., `a0` has value `14` and address `16`).

The IPython console shows the execution of `runfile` and the resulting output:

```
In [28]: runfile('C:/Users/antho/Desktop/backup_files/dev/untitled0.py', wdir='C:/Users/antho/Desktop/backup_files/dev')
a0      14      16
a1      4      16
a2     14      16
a3      8      16
a4      6      16
In [29]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 22, Column: 1, Memory: 83 %.

Python File Handling (4)

The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script named 'untitled0.py' with the following code:

```
1 # File Handling
2
3 file_object = open("ibm02.nodes")
4
5 line1 = file_object.readline()
6 print(type(line1))
7 print(line1)
8
9 line2 = file_object.readline().split()
10 print(line2)
11 print(type(line2))
12 print(line2[0])
13 print(line2[1])
14
15 file_object.close()
16
17
18
```

The variable explorer on the right shows a table of variables:

1	a0	14	16
2	a1	4	16
3	a2	14	16
4	a3	8	16
5	a4	6	16
6	a5	4	16
7	a6	6	16
8	a7	14	16
9	a8	4	16
10	a9	14	16
11	a10	6	16
12	a11	6	16
13	a12	4	16
14	a13	6	16
15	a14	16	16
16	a15	8	16
17	a16	14	16
18	a17	6	16
19	a18	6	16
20	a19	14	16
21	a20	4	16
22	a21	6	16
23	a22	14	16
24	a23	10	16
25	a24	8	16
26	a25	6	16
27	a26	6	16
28	a27	6	16
29	a28	16	16
30	a29	14	16
31	a30	6	16

The Python console shows the output of the script:

```
In [34]: runfile('C:/Users/antho/Desktop/backup_files/dev/untitled0.py', wdir='C:/Users/antho/Desktop/backup_files/dev')
<class 'str'>
    a0          14          16

['a1', '4', '16']
<class 'list'>
a1
4

In [35]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 16, Column: 1, Memory: 87 %.

Python File Handling (5)

The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script named `ibm02.nodes` with the following code:

```
1 # File Handling
2
3 file_object = open("ibm02.nodes")
4
5 all_lines = file_object.readlines()
6
7 for line in all_lines:
8     if line.strip()[0][0] == "a":
9         print(line)
10
11 file_object.close()
12
```

The output window shows the results of the script execution, displaying a list of lines from the file `ibm02.nodes` that start with the letter 'a'. The output is as follows:

19325	a19324	6	16
19326	a19325	14	16
19327	a19326	6	16
19328	a19327	4	16
19329	a19328	14	16
19330	a19329	14	16
19331	a19330	14	16
19332	a19331	6	16
19333	a19332	14	16
19334	a19333	10	16
19335	a19334	10	16
19336	a19335	4	16
19337	a19336	8	16
19338	a19337	6	16
19339	a19338	4	16
19340	a19339	16	16
19341	a19340	4	16
19342	a19341	2	16
19343	p1	1	1 t
19344	p2	1	1 t
19345	p3	1	1 t
19346	p4	1	1 t
19347	p5	1	1 t
19348	p6	1	1 t
19349	p7	1	1 t
19350	p8	1	1 t
19351	p9	1	1 t
19352	p10	1	1 t
19353	p11	1	1 t
19354	p12	1	1 t
19355	p13	1	1 t

The bottom status bar indicates: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 1, Column: 23, Memory: 84 %.

Python File Handling (6)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script for file handling. The script opens a file named 'temp.txt' in write mode, writes the text 'user generated numbers' followed by a newline, and then prompts the user to enter two numbers (101 and 55) which are also written to the file. The file is then closed.

```
1 # File Handling
2
3 # Writing data to a file can be achieved with
4 # the usage of open().
5
6 # open() attributes:
7 # mode      : r/w/a/x/r+/w+/a+/x+
8 # buffering : how should intermediate memory be used
9 # encoding  : file encoding
10 # errors    : error manipulation
11 # newline   : new line character manipulation
12 # closefd   : define file closing
13
14 file_object = open("temp.txt", "w")
15 file_object.write("user generated numbers\n")
16
17 for i in range(2):
18     tmp_val = input("type a number: ")
19     file_object.write("\t" + tmp_val + "\n")
20
21 file_object.close()
22
23
```

The IPython console shows the execution of the script, including the user input and the resulting output:

```
In [53]: runfile('C:/Users/antho/Desktop/backup_files/dev/untitled0.py', wdir='C:/Users/antho/Desktop/backup_files/dev')
type a number: 101
type a number: 55
In [54]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 22, Column: 1, Memory: 85 %.

Python File Handling (7)

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named 'untitled0.py' with the following code:

```
1 # File Handling & Folder Handling
2
3 # Homework:
4 # - writelines()
5 # - csv module
6 # - json module
7
8 import os
9
10 print("C:\\Users\\antho\\Desktop\\backup_files\\dev\\ibm01.nets")
11
12 file_object = open("C:\\Users\\antho\\Desktop\\backup_files\\dev\\ibm01.nets")
13 file_object.close()
14
15 # Get current working directory
16 print(os.getcwd())
17
18 # A list containing every file in the current directory
19 print(os.listdir()[0:10])
20
21 # Creating a directory
22 os.mkdir("temp_dir")
23
24 # Renaming a file
25 os.rename("ibm_approach_1.py", "ibm_approach_2.py")
26
27 # Deleting a file
28 os.remove("ibm_approach_2.py")
29
```

The File Explorer window on the right shows the directory structure of the current working directory, 'C:\\Users\\antho\\Desktop\\backup_files\\dev'. It contains a table of files and folders:

Name	Size	Type	Date Modified
> _pycache_		File Folder	4/6/2020 11:11 A...
> temp_dir		File Folder	4/6/2020 8:58 PM
fixing_tetris_notes.txt	259 KB	txt File	4/5/2020 11:35 A...
fixing_tetris.py	5 KB	py File	3/24/2020 5:29 PM
ibm_backup_1.py	260 KB	py File	3/30/2020 9:47 A...
ibm_backup.py	258 KB	py File	4/3/2020 3:37 PM

The Python console window shows the output of the script execution:

```
In [70]:
In [70]: runfile('C:/Users/antho/Desktop/backup_files/dev/untitled0.py', wdir='C:/Users/antho/Desktop/backup_files/dev')
C:\Users\antho\Desktop\backup_files\dev\ibm01.nets
C:\Users\antho\Desktop\backup_files\dev
['desktop.ini', 'fixing_tetris.py', 'fixing_tetris_notes.txt', 'ibm01.nets', 'ibm01.nodes', 'ibm01.pl', 'ibm01.py', 'ibm01.scl', 'ibm02.nets', 'ibm02.nodes']
In [71]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 29, Column: 1, Memory: 87 %.

Python Code “Quality” (1)

- Good Code:
 - Does what is supposed to do.
 - Does not contain bugs, errors, logic errors, etc.
 - Readable
 - Maintainable
 - Extensible
- Style Guides:
 - PEP8 (<https://pep8.org/>) & PEP257 (<https://www.python.org/dev/peps/pep-0257/>)
- Linters: The “annoying” underlines generated in most IDEs

Python Code “Quality” (2)

- Linters (errors, potential errors, code patterns, style):
 - PyLint
 - PyFlakes
 - Bandit
 - MyPy
 -
- You should find the appropriate plug-in for the IDE of your choice....

“Laws”, “Rules” & “Principles” related to Software Development/Coding



- Murphy's Law: “If something can go wrong, it will”
- Brook's Law: “Adding manpower to a late software project makes it later”
- Conway's Law: “Any piece of software reflects the organizational structure that produced it”
- Linus's Law: “Given enough eyeballs, all bugs are shallow”
- Ninety-ninety rule: “The first 90% of the code takes 10% of the time, the remaining 10% takes the other 90% of the time”
- Knuth's optimization principle: “Premature optimization is the root of all evil”

Επαναληπτικές Ασκήσεις

1 - Παρουσιάστε ένα απλό παράδειγμα για κάθε έναν από τους αριθμητικούς τελεστές της γλώσσας.

2 - Δημιουργήστε δύο μεταβλητές με λάθος ονομασία και παρατηρήστε τον τρόπο με τον οποίο παράγονται ειδοποιήσεις στην κονσόλα.

3 - Γράψτε ένα μικρό παράδειγμα χρήσης της εντολής `input`.

4 - Ποιά θα είναι η τιμή της μεταβλητής `a` μετά την εκτέλεση των παρακάτω αναθέσεων;

- $a = 7 - 6 / 4$
- $a = a / 2 - a / 3 - a / 4$
- $a = 1 - 1 / 5 - 5 / 8 - 2 + 11$

5 - Γράψτε τις εντολές εκχώρησης ώστε η μεταβλητή `a` να πάρει τις παρακάτω τιμές:

- Τον μέσο όρο των μεταβλητών `b`, `c`, `d`, `e`.
- Το υπόλοιπο της διαίρεσης του `b` με το `c`, προσαυξημένο κατά `d`.
- Την απόλυτη τιμή της διαφοράς των `c` και `d`.

Επαναληπτικές Ασκήσεις

6 - Γράψτε ένα πρόγραμμα το οποίο δέχεται έναν πενταψήφιο αριθμό ως είσοδο (δεν χρειάζεται να ελεγχθεί το πλήθος των ψηφίων) και επιστρέφει το άθροισμα των ψηφίων του και τον αριθμό ανεστραμμένο.

7 - Γράψτε ένα πρόγραμμα το οποίο υπολογίζει την Ευκλείδεια απόσταση δύο σημείων.

8 - Γράψτε ένα πρόγραμμα το οποίο δέχεται τρεις αριθμούς και επιστρέφει τον μέγιστο, χωρίς τη χρήση της συνάρτησης `max`.

9 - Γράψτε ένα πρόγραμμα το οποίο παράγει τα 6 νούμερα του λόττο (από 1 έως 49). Ποιό είναι ένα πιθανό πρόβλημα που μπορεί να προκύψει από την εκτέλεση του κώδικα;

10 - Να γραφεί ένα πρόγραμμα το οποίο θα υπολογίζει την ακόλουθη συνάρτηση:

- $f(x,y) =$
 - $x - y, \quad x \geq 0, y \geq 0$
 - $x^4 - y^3, \quad x \geq 0, y < 0$
 - $x - |y|, \quad x < 0, y < 0$

Επαναληπτικές Ασκήσεις

11 - Υλοποιήστε το παιχνίδι πέτρα-ψαλίδι-χαρτί. Οι παίκτες που συμμετέχουν είστε εσείς και ο υπολογιστής (οι επιλογές του υπολογιστή πρέπει να γίνονται με τυχαίο τρόπο). Ο αριθμός των γύρων που θα κρίνουν τον τελικό νικητή πρέπει να δίνεται στην αρχή της εκτέλεσης του προγράμματος από το χρήστη.

12 - Ένας βιομηχανικός ανελκυστήρας μπορεί να δεχθεί μέχρι 10 πακέτα και έως 1000 κιλά βάρος. Να γραφεί κατάλληλο πρόγραμμα το οποίο θα δέχεται κατ'επανάληψη το βάρος κάθε πακέτου και κατά την ολοκλήρωσή του θα εμφανίζει τον συνολικό αριθμό πακέτων και το συνολικό βάρος που έχει μεταφερθεί.

13 - Να γραφεί ένα πρόγραμμα το οποίο θα δέχεται έναν αριθμό n μεγαλύτερο του 10 στην είσοδό του και θα εμφανίζει όλους τους αριθμούς από το 2 μέχρι το n που είναι πρώτοι.

Επαναληπτικές Ασκήσεις

14 - Να γραφεί ένα πρόγραμμα το οποίο θα ζητάει το password του χρήστη προτού προχωρήσει σε κάποια άλλη λειτουργία. Σε περίπτωση τριών αποτυχημένων προσπαθειών το πρόγραμμα θα "κλειδώνει" για ένα λεπτό. (tip: χρησιμοποιήστε την συνάρτηση sleep του module time).

15 - Να γραφεί ένα πρόγραμμα το οποίο θα παράγει μια λίστα που θα περιλαμβάνει τις τετραγωνικές ρίζες των άρτιων αριθμών μιας λίστας.

16 - Να γραφεί ένα πρόγραμμα το οποίο θα διαβάζει μια λίστα με αριθμούς από το 1 έως το 100 (επιτρέπονται τα διπλότυπα) και θα επιστρέφει μια νέα λίστα με τη συχνότητα εμφάνισης των αριθμών.

17 - Να γραφεί ένα πρόγραμμα το οποίο θα εμφανίζει την συχνότητα εμφάνισης του κάθε χαρακτήρα ενός κειμένου που θα δίνεται από τον χρήστη.

Επαναληπτικές Ασκήσεις

18 - Διαβάστε και υλοποιήστε ένα παράδειγμα χρήσης του αλγορίθμου κρυπτογράφησης του Καίσαρα.

19 - Να γραφεί ένα πρόγραμμα το οποίο θα αφαιρεί από ένα κείμενο όλα τα tabs και τα κενά και θα τα αντικαθιστά με underscores.

20 - Να γραφεί ένα πρόγραμμα το οποίο θα επιστρέφει τον αριθμό συμφώνων και φωνηέντων ενός κειμένου.

21 - Να γραφεί ένα πρόγραμμα το οποίο θα δέχεται έναν κωδικό από τον χρήστη και θα κρίνει την δυνατότητα χρήσης του βάσει των παρακάτω προϋποθέσεων:

- πρέπει να αποτελείται από 8 έως 11 χαρακτήρες
- πρέπει να περιλαμβάνει
 - τουλάχιστον ένα κεφαλαίο γράμμα
 - έναν αριθμό και ένα special character

Επαναληπτικές Ασκήσεις

22 - Να γραφεί ένα πρόγραμμα το οποίο θα διαβάσει ένα κείμενο που δίνεται από τον χρήστη και θα επιστρέφει όλους τους διαφορετικούς χαρακτήρες που εμφανίζονται μέσα σε αυτό.

23 - Να γραφεί ένα πρόγραμμα το οποίο θα δέχεται ένα κείμενο από τον χρήστη και θα δημιουργεί ένα λεξικό με κλειδιά τα γράμματα του κειμένου και τιμές τον αριθμό εμφάνισής τους.

24 - Να γραφεί ένα πρόγραμμα το οποίο δημιουργεί ένα λεξικό βάσει των πληροφοριών που εισάγει ο χρήστης και αφορούν κάποιο προϊόν (κωδικός προϊόντος και τιμή). Η διαδικασία θα ολοκληρώνεται όταν ο χρήστης δώσει αρνητική τιμή για κάποιο προϊόν. Μετά το πέρας της παραπάνω διαδικασίας θα εμφανίζεται στην οθόνη το προϊόν με τη μεγαλύτερη τιμή, το προϊόν με τη μικρότερη τιμή και η μέση τιμή προϊόντος.

Επαναληπτικές Ασκήσεις

25 - Να γραφεί μια συνάρτηση που θα παράγει τυχαίους αριθμούς από το 1 έως και το 100, φροντίζοντας να μην επαναλαμβάνεται ο ίδιος αριθμός στην ίδια κλήρωση.

26 - Να γραφεί μια συνάρτηση που θα δέχεται ένα αλφαριθμητικό και θα επιστρέφει ένα λεξικό με κλειδιά τους χαρακτήρες του κειμένου και τιμές το πλήθος εμφανίσεων του κάθε χαρακτήρα.

27 - Να γραφεί ο αλγόριθμος σειριακής αναζήτησης με την μορφή συνάρτησης.

28 - Να γραφεί μια συνάρτηση που θα ελέγχει κατά πόσο ένας αριθμός είναι τέλειος.