Type Conversion:

```
1. str(): Converts to a string.
num = 10
num_str = str(num)

2. int(): Converts to an integer.
num_str = "10"
num = int(num_str)

3. float(): Converts to a float.
num_str = "10.5"
num = float(num_str)

4. list(): Converts to a list.
my_string = "hello"
my_list = list(my_string)
```

Data Structures:

```
List Methods:
1. append(), extend(), insert(), remove(), pop(), index(), count(),
sort(), reverse()
my_list = [1, 2, 3]
my_list.append(4)

Tuple Methods:
1 index(), count()
my_tuple = (1, 2, 3)
index = my_tuple.index(2)

Set Methods:
1. add(), remove(), union(), intersection(), difference(),
symmetric_difference()
set1 = {1, 2, 3}
set2 = {2, 3, 4}
union_set = set1.union(set2)
```

Dictionary Methods:

```
1. keys(), values(), items(), get(), pop(), update()
my_dict = {'a': 1, 'b': 2}
keys = my_dict.keys()
```

String Manipulation:

```
1. split(): Splits a string into a list.
my_string = "Hello, World"
split_string = my_string.split(',')

2. join(): Joins elements of an iterable with a string.
my_list = ['Hello', 'World']
joined_string = ' '.join(my_list)

3. strip(): Removes leading and trailing whitespace.
my_string = "   Hello   "
stripped_string = my_string.strip()
```

List Manipulation:

```
1. append(): Adds an element to the end of the list.
my_list = [1, 2, 3]
my_list.append(4)

2. extend(): Extends a list by appending elements from another list.
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list1.extend(list2)

3. pop(): Removes and returns an element at a given index.
my_list = [1, 2, 3]
popped_element = my_list.pop(1)
```

Dictionary Manipulation:

1. keys(): Returns a list of dictionary keys.

```
my_dict = {'a': 1, 'b': 2}
keys = my_dict.keys()
```

2. values(): Returns a list of dictionary values.

```
my_dict = {'a': 1, 'b': 2}
values = my_dict.values()
```

3. items(): Returns a list of key-value pairs in a dictionary.

```
my_dict = {'a': 1, 'b': 2}
items = my_dict.items()
```

Control Flow:

1. if, elif, else: Conditional statements for decision making.

```
x = 10
if x > 5:
    print("x is greater than 5")
else:
    print("x is less than or equal to 5")
```

2. for loop: Iterates over a sequence.

```
for i in range(5):
    print(i)
```

3. while loop: Executes a block of code as long as a condition is true.

```
x = 0
while x < 5:
    print(x)
x += 1
```

File Handling:

```
1. open(): Opens a file.
file = open('example.txt', 'r')

2. read(): Reads the content of the file.
content = file.read()

3. write(): Writes content to the file.
file.write('Hello, this is a test.')
```

Math Operations:

```
1. sum(): Returns the sum of elements in an iterable.
my_list = [1, 2, 3]
total = sum(my_list)

2. min(), max(): Returns the minimum or maximum value in an iterable.
my_list = [1, 2, 3]
min_value = min(my_list)
max_value = max(my_list)
```

Boolean Operations:

```
1. and, or, not: Logical operators for boolean expressions.

x = True
y = False
result = x and y
```

Exception Handling:

1. try, except: Catches and handles exceptions.

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
```

Object-oriented Programming:

1. class: Defines a new class.

```
class MyClass:
    def __init__(self, x):
        self.x = x
```

2. init(): Initializes an object.

```
obj = MyClass(5)
```

3. self: Represents the instance of the class.

```
class MyClass:
    def __init__(self, x):
        self.x = x
```

Module-related:

1. import: Imports a module.

```
import math
```

2. from...import: Imports specific attributes or functions from a module.

```
from math import sqrt
```

Functional Programming:

```
1. map(): Applies a function to all items in an input list.
def square(x):
    return x * x

numbers = [1, 2, 3, 4]
squared_numbers = list(map(square, numbers))

2. filter(): Filters elements from an iterable based on a function.
def is_even(x):
    return x % 2 == 0

numbers = [1, 2, 3, 4, 5, 6]
even_numbers = list(filter(is_even, numbers))
```