Concepts and Principles
Sequence
Selection
Repetition (loops)
Modularisation (functional decomposition)

Programming Artefacts
Variable
Function
main() function
if and match
while and for
lists and dicts
Custom data types (data classes)

Sequence

A sequence in programming is a way in which it will run or execute, be it from top to bottom, left to right, aesc or desc.

Example:

1. odd_nums = [1,3,5,7,9]
2. print(“Hello”)
3. print(“Everyone”)
4. print(“!”)
5. print(f”{odd_nums}”)

In this example, python runs the code from top to bottom, where it first assigns [1,3,5,7,9] in the variable odd_nums in line 1. It prints Hello in line 2. It prints Everyone in line 3, It prints 1 in line 4 and finally prints the value stored in odd_nums which is [1,3,5,7,9] in line 5.

Selection

A selection is programming is a decision of which line of code should run under a given condition.

Example:

1. if 10%2 == 0:
    a. print (“given number is even”)
2. else
    a. print (“given number is odd”)

3. num = 10
4. match num:
    a. case “5”

        print (“Better luck next time!”)

```
    b. case “10”
        i. print (“found 10!”)
```

In line 1, it checks if 10 divided by 2 has a remainder using the modulus operator (%). Since 10 % 2 equals 0, the condition is true. Which makes the program to select 1.a So in line 1.a, it prints “given number is even”. The else block in line 2 and line 2.a is skipped because the if condition was true. In line 4, it uses a match statement to check what value num holds. In line 4.a, it checks if num is 5, which is false, so line 5.a is skipped. In line 5.b, it checks if num is 10, which is true, so the program selects 4.b.i and prints “found 10!”.

Repetition (loops)

Repetition is a technique of continuous execution of a line or a block of code until the range or condition is changed.

Example:

```
1. for i in range (1,10)
    a. print (i)

2. num = 1
3. while num>10:
    a. print(“True”)
    b. num += 1
```

In line 1 the program runs from 1 to 9(which is repetition) and prints each number using line 1.a. The while loop in line 3 checks if num > 10(also a repetition), but since num is 1, the condition is false. So the while loop is skipped and nothing is printed from it and the repetition never happens.

Modularisation(functional decomposition)

Modularisation in programming is a structure that is followed to break complex programs into meaningful functions which improve code readability, scalability and maintenance.

Example:

```
1. def add_nums(int a, int b) -> int
    a. return a+b

2. add_nums(6,5)
```

In this example, modularisation is used by creating a function add_nums that takes two numbers and returns their sum. It shows that the task can be divided into chunks to use it or in programming language “call” it from any part of the program. The function is called with 6 and 5 in line 2. This makes the code reusable and easier to manage.

Variables

Variables in programming are like containers which store the reference for the data stored in the memory.

Example:

```
name: str = “Sachin”
```

This *name* variable will store the reference for the string Sachin stored in the memory.

Function

Functions in programming are reusable blocks of code that can be defined and used multiple times in different parts of the program. It helps to break complex concepts into meaningful blocks which improve code readability, scalability and maintenance.

Example:

```
def add_nums(int a, int b) -> int
        return a+b

add_nums(6,5)
```

In this example, function is used by creating a function add_nums that takes two numbers and returns their sum. It shows that the task can be divided into chunks to use it or in programming language “call” it from any part of the program. The function is called with 6 and 5 in line 2. This makes the code reusable and easier to manage.

main() function

main() function is literally the “main” function of in python as the name suggests, which runs when the program is executed, unlike other functions it is called differently, like:

```
if __name__ == "__main__":
        main()
```

if and match

if in programming comes under selection where it matches the code conditionally, when one or the other condition is matched. It is mainly used where there are minimal cases.

Example:

```
1. if 10%2 == 0:
      a. print ("given number is even")
2. else
      a. print ("given number is odd")
```

In line 1, it checks if 10 divided by 2 has a remainder using the modulus operator (%). Since 10 % 2 equals 0, the condition is true. Which makes the program to select 1.a So in line 1.a, it prints "given number is even". The else block in line 2 and line 2.a is skipped because the if condition was true.

match in programming also comes under selection where it matches the code as a pattern, when one or the other pattern is matched. It is mainly used where there are several cases.

Example:

```
1. num = 10
2. match num:
      a. case "5"
                    print ("Better luck next time!")
      b. case "10"
            i.   print ("found 10!")
```

In line 2, it uses a match statement to check what value num holds. In line 2.a, it checks if num is 5, which is false, so line 2.a is skipped. In line 2.b, it checks if num is 10, which is true, so the program selects 2.b.i and prints "found 10!".

while and for

while and for in python comes under repetition where they are used to continuously execute a line or a block of code until the range or condition is changed.

Example:

```
1. for i in range (1,10)
      a. print (i)

2. num = 1
3. while num>10:
      a. print("True")
      b. num += 1
```

In line 1 the program runs from 1 to 9(which is a for loop) and prints each number using line 1.a. The while loop in line 3 checks if num > 10(which is a while loop), but since num is 1, the condition is false. So the while loop is skipped and nothing is printed from it and the repetition never happens.

lists and dicts

lists in python are ordered collections which are used to store data in it where its values are indexed by the indexes staring at 0.

Example:

```
odd_nums = [1,3,5,7,9]
```

In this example, odd_nums is a list that stores the odd numbers 1, 3, 5, 7, and 9. The values are ordered, and each item has an index starting from 0. For example, odd_nums[0] will give 1, and odd_nums[2] will give 5.

dicts in python are key-value type storage where the key acts as indexes that are connected with its value. Keys can be initialised by the user with their values.

Example:

```
my_details = {“name”: “Sachin”, “ID”: 689206}
```

In this example, my_details is a dictionary with two key-value pairs. "name" is the key for the value "Sachin", and "ID" is the key for the value 689206. This structure allows us to access values using their keys, like my_details["name"] will give "Sachin".

Custom data types (data classes)
Custom data types (data classes) are used when we need to capsulise related data in one object, which then can be used in multiple parts of the program just by calling the data class with its related fields.

Example:

```
@dataclass
class University:
  name: str
  est: int
  rank: int

my_uni = University("UTAS", 1980, 1)
```

In this example, a @dataclass defines a reusable University class with attributes: name, est, and rank. The my_uni object is created with the values "UTAS", 1980, and 1. This makes the structure reusable, so you can easily create more university objects with the same format.