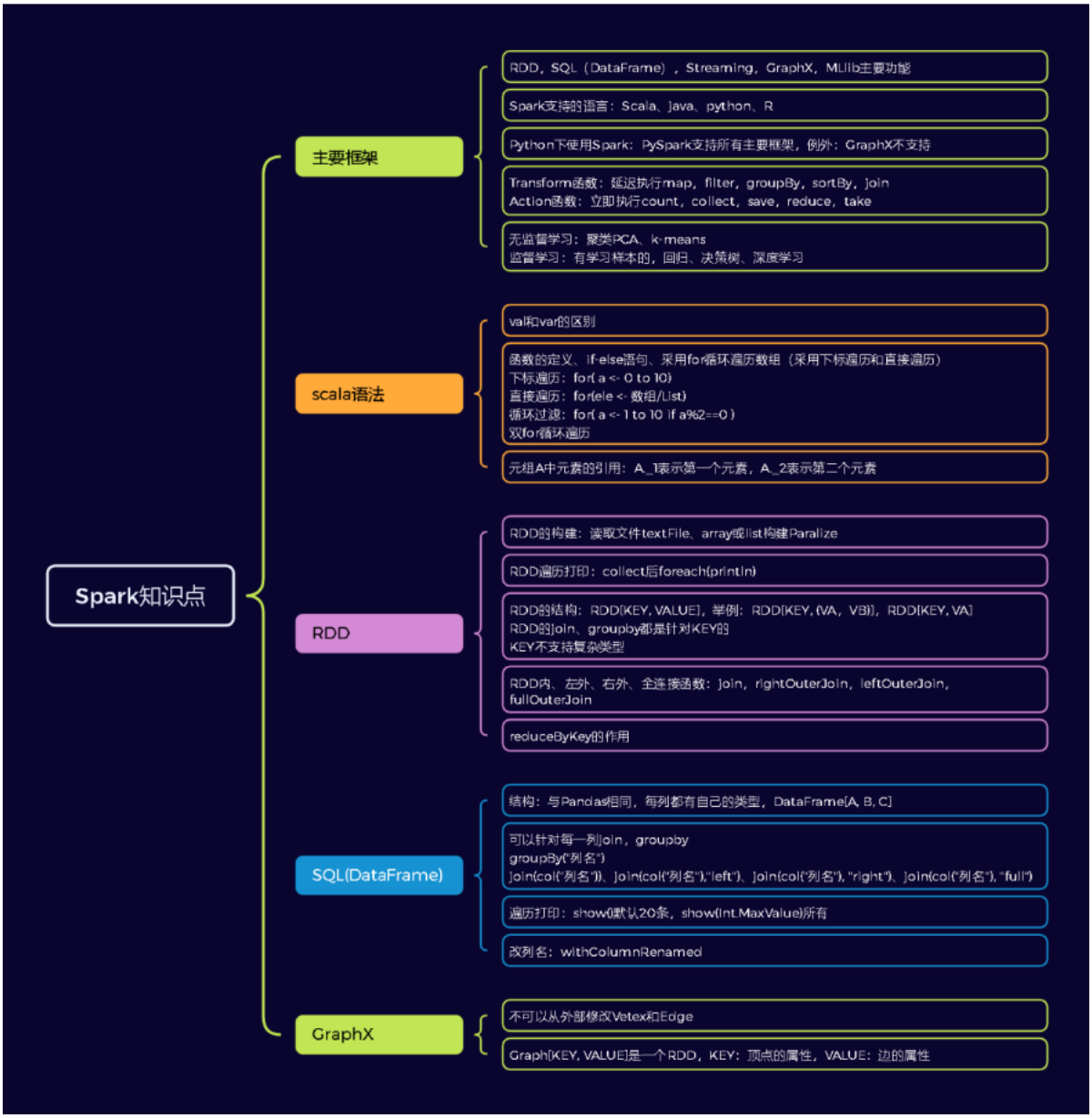


1.填空（4道题 15分

函数、返回值、变量定义 类/构造

> 1、构建一个类，输入一个数组（10个数）
>
> | | | | | | | | | | |
> |---|---|---|---|---|---|---|---|---|---|
> | 70 | 19 | 69 | 91 | 58 | 85 | 15 | 89 | 75 | 27 |
>
> 2、在构造函数中遍历打印该数组
>
> 3、新建一个函数找到该数组中最大的数，作为函数返回值返回（采用下标遍历该数组）
>
> 4、在main函数中调用该函数，并对函数返回的最大值进行打印。

```
object TestMax {
def main(args: Array[String]): Unit = {
 //变量的定义：  名称:类型=值
```

```
 var arr: Array[Int] = Array(70, 19, 69, 91, 58, 85, 15, 89, 75, 27)
 //新建一个类的实例
 var getmax: GetMax = new GetMax(arr)

 println(getmax.findMax())
}
}

//新建一个类
class GetMax(arrIn: Array[Int]) {
var arr: Array[Int] = arrIn
//用for直接遍历数组
for (element <- arr) {
 println(element)
}

def findMax(a:Int=0): Int = {
 var maxValue = Int.MinValue
 //for用下标遍历数组
 for (i <- 0 until arr.length) {
   //if语句
   if (arr(i) > maxValue) {
     //array（标号）引用数组的元素
     maxValue = arr(i)
   }
 }
 maxValue
}
}
```

1、新建一个函数实现两个整数相乘，并返回结果

2、新建一个函数实现数组的遍历，打印数组中的元素

```
var arr:Array[Int] = Array(70,19,69,91,58,85,15,89,75,27)
```

3、新建一个函数对输入的两个整数比较大小，返回大的那个

4、新建一个类Basic，类中包含以上函数

5、在object的main函数中，新建Basic类的实例，并调用以上函数，打印函数返回值

```
object TestBasic {
def main(args: Array[String]): Unit = {
 var b: Basic = new Basic()
 println(b.multiply(1, 2))
 b.showArr()
 println(b.large(10,100))
}
}
class Basic {
def multiply(a: Int, b: Int): Int = {
 return a * b
}
```

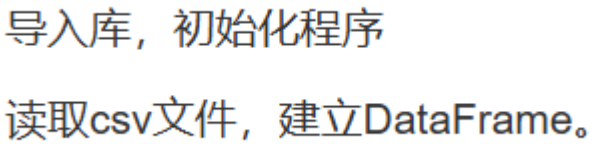
```
def showArr(): Unit = {
 var arr: Array[Int] = Array(70, 19, 69, 91, 58, 85, 15, 89, 75, 27)
 for (element <- arr) {
   println(element)
 }
}
def large(a: Int, b: Int): Int = {
 if (a > b)
   return a
 else
   return b
}
}
```

2.选择题（20道题 40分

20题 40分

3.编程题（45分

1.DataFrame练习



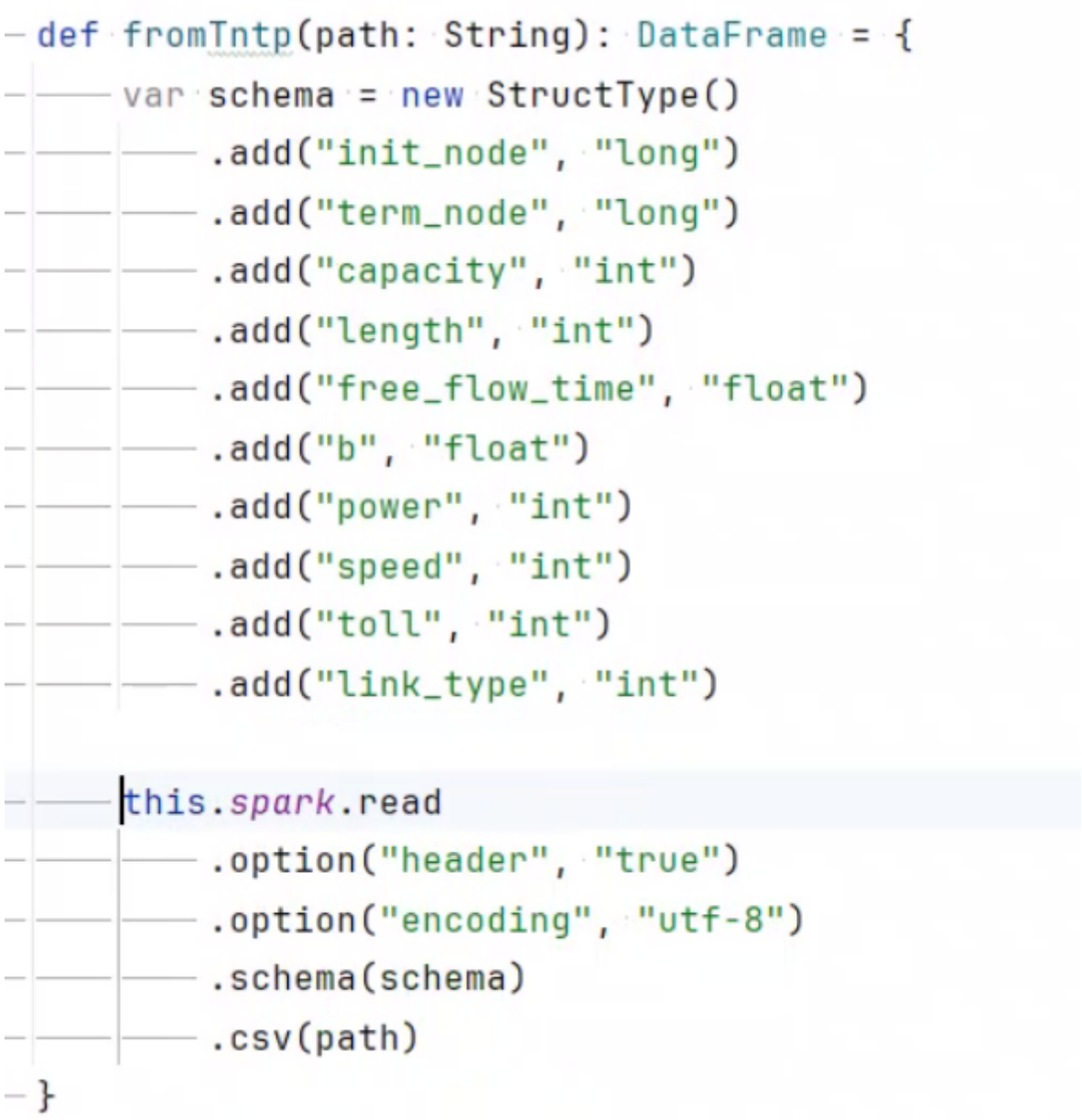
提示内容：

```
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.{DataFrame, SparkSession}

object TestAnaheim {
val spark=SparkSession
 .builder()  //使用SparkSession的builder方法创建SparkSession对象
 .appName("SparkSQL") //appName等效于SparkContext的setAppName方法
 .master("local[*]")  //master等效于SparkContext的setMaster方法
 .getOrCreate() //如果对象已存在则使用它否则创建它
val sc = spark.sparkContext
sc.setLogLevel("WARN")
val fileDir = "F:\testscala\testscala\src\TestAnaheim\"
```

```
def main(args: Array[String]): Unit = {}
}
```

读文件提示代码：




```
var schema = new StructType()
  .add("init_node", "long")
this.spark.read
  .option("header", "true")
  .option("encoding", "utf-8")
  .schema(schema)
  .csv(path)
```

代码：

```
package Test
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.{DataFrame, SparkSession, types}
```

```
object Test3 {
  val spark=SparkSession
    .builder()   //使用SparkSession的builder方法创建SparkSession对象
    .appName("SparkSQL")  //appName等效于SparkContext的setAppName方法
    .master("local[*]")   //master等效于SparkContext的setMaster方法
    .getOrCreate()  //如果对象已存在则使用它否则创建它
  val sc = spark.sparkContext
  sc.setLogLevel("WARN")
  val fileDir = "A:\\A_2024-2025last\\Spark大数据技术与应用\\spark复习"

  def main(args: Array[String]): Unit = {
    var tntp = this.fromTntp(this.fileDir+"Anaheim_net.csv")
    tntp.show()
  }

  def fromTntp(path: String): DataFrame = {

    var verticeSchema = new StructType()
    var schema = new types.StructType()
      .add("init_node", "long")
      .add("term_node", "long")
      .add("capacity", "int")
      .add("length", "int")
      .add("free_flow_time", "float")
      .add("b", "float")
      .add("power", "int")
      .add("speed", "int")
      .add("toll", "int")
      .add("link_type", "int")

    this.spark.read
      .option("header", "true")
      .option("encoding", "utf-8")
      .schema(schema)
      .csv(path)
  }
}
```

2.对DataFrame进行检索

在dataframe中查找free_flow_time大于1.5的节点，用“起始节点”、“终结点”、“自由流时间”的方式进行显示；

1、采用sql语句进行检索

2、采用API进行检索

提示内容：

> createOrReplaceTempView
> spark.sql(sqlstr)

> where/filter

在 dataframe 中查找 free_flow_time 大于 1.5 的节点，用“起始节点”、“终结点”、“自由流时间”的方式进行显示；
1、采用sql语句进行检索
2、采用API进行检索

代码：

```
package Test
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.{DataFrame, SparkSession, types}
object Test3 {
  val spark=SparkSession
    .builder()   //使用SparkSession的builder方法创建SparkSession对象
    .appName("SparkSQL")  //appName等效于SparkContext的setAppName方法
    .master("local[*]")   //master等效于SparkContext的setMaster方法
    .getOrCreate()  //如果对象已存在则使用它否则创建它
  val sc = spark.sparkContext
  sc.setLogLevel("WARN")
  val fileDir = "A:\\A_2024-2025last\\Spark大数据技术与应用\\spark复习"

  def main(args: Array[String]): Unit = {
    var tntp = this.fromTntp(this.fileDir+"Anaheim_net.csv")
    tntp.show()
  }

  def fromTntp(path: String): DataFrame = {

    var verticeSchema = new StructType()
    var schema = new types.StructType()
      .add("init_node", "long")
      .add("term_node", "long")
      .add("capacity", "int")
      .add("length", "int")
      .add("free_flow_time", "float")
      .add("b", "float")
      .add("power", "int")
      .add("speed", "int")
      .add("toll", "int")
      .add("link_type", "int")

    var df =this.spark.read
      .option("header", "true")
      .option("encoding", "utf-8")
      .schema(schema)
      .csv(path)
    //问题二
    df.createOrReplaceTempView("netview")
    var ret = this.spark.sql("select init_node as `起始节点`,term_node as `终结点
`,free_flow_time as `自由流时间` from netview where free_flow_time>1.5;")

    //问题三
    df.where("free_flow_time>1.5")
```

```
      .selectExpr("init_node as `起始节点`", "term_node as `终结点`",
"free_flow_time as `自由流时间`")

  }
}
```

3.完成任务

> 1、根据free_flow_time对数据从高到低进行排序，按照“开始节点”，“终结点”，“通行时间”的方式进行显示。
>
> 2、显示通行时间最长的十个路段信息，按照“开始节点”，“终结点”，“通行时间”的方式进行显示；
>
> 3、显示不同speed条件下的平均通行时间，按照“速度”、“平均通行时间”的方式进行显示；
>
> 4、显示不同speed条件的最大和最小通行时间，按照“速度”、“最大通行时间”、“最小通行时间”的方式进行显示。

```
 //问题四：根据free_flow_time对数据从高到低进行排序，按照“开始节点”，“终结点”，“通行时间”的方式进行显示
        df.sort(desc("free_flow_time"))
            .selectExpr("init_node as `起始节点`","term_node as `终结点
`","free_flow_time as `自由流时间`")


        //问题五：显示通行时间最长的十个路段信息，按照“开始节点”，“终结点”，“通行时间”的方式进行显示；
        df.sort(desc("free_flow_time"))
            .selectExpr("init_node as `起始节点`","term_node as `终结点
`","free_flow_time as `自由流时间`")
            .take(10)
            //.foreach(println)


        //问题六：显示不同speed条件下的平均通行时间，按照“速度”、“平均通行时间”的方式进行显示；
        df.groupBy("speed")
            .avg("free_flow_time")
            .selectExpr("speed as `速度`","`avg(free_flow_time)` as `平均通行时间`")

        //问题七：显示不同speed条件下的最大和最小通行时间，按照“速度”、“最大通行时间”、“最小通行时间”的方式进行显示。
        df.groupBy("speed").agg(max("free_flow_time"),min("free_flow_time"))
            .selectExpr("speed as `速度`","`max(free_flow_time)` as `最大通行时间
`","`min(free_flow_time)` as `最小通行时间`")
            .show()
```