Operating Systems

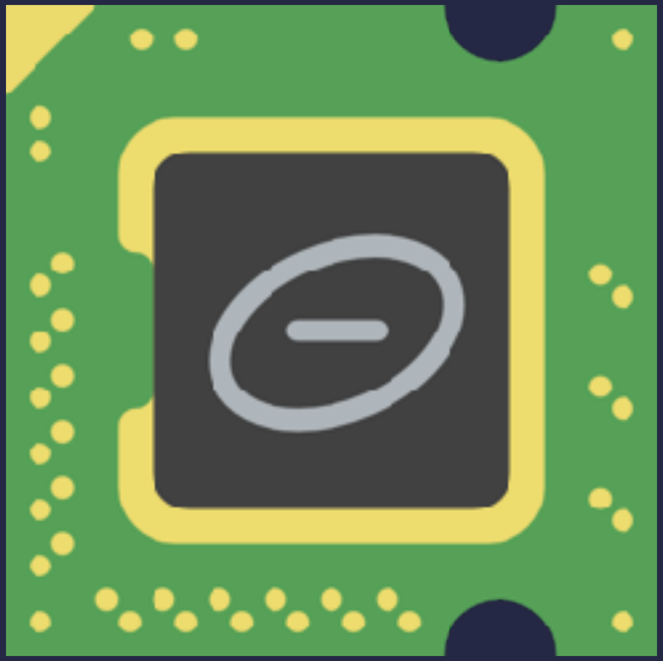
CPU Scheduling Simulation

Department of Computer Science
Vincent Mary School of Science and Technology

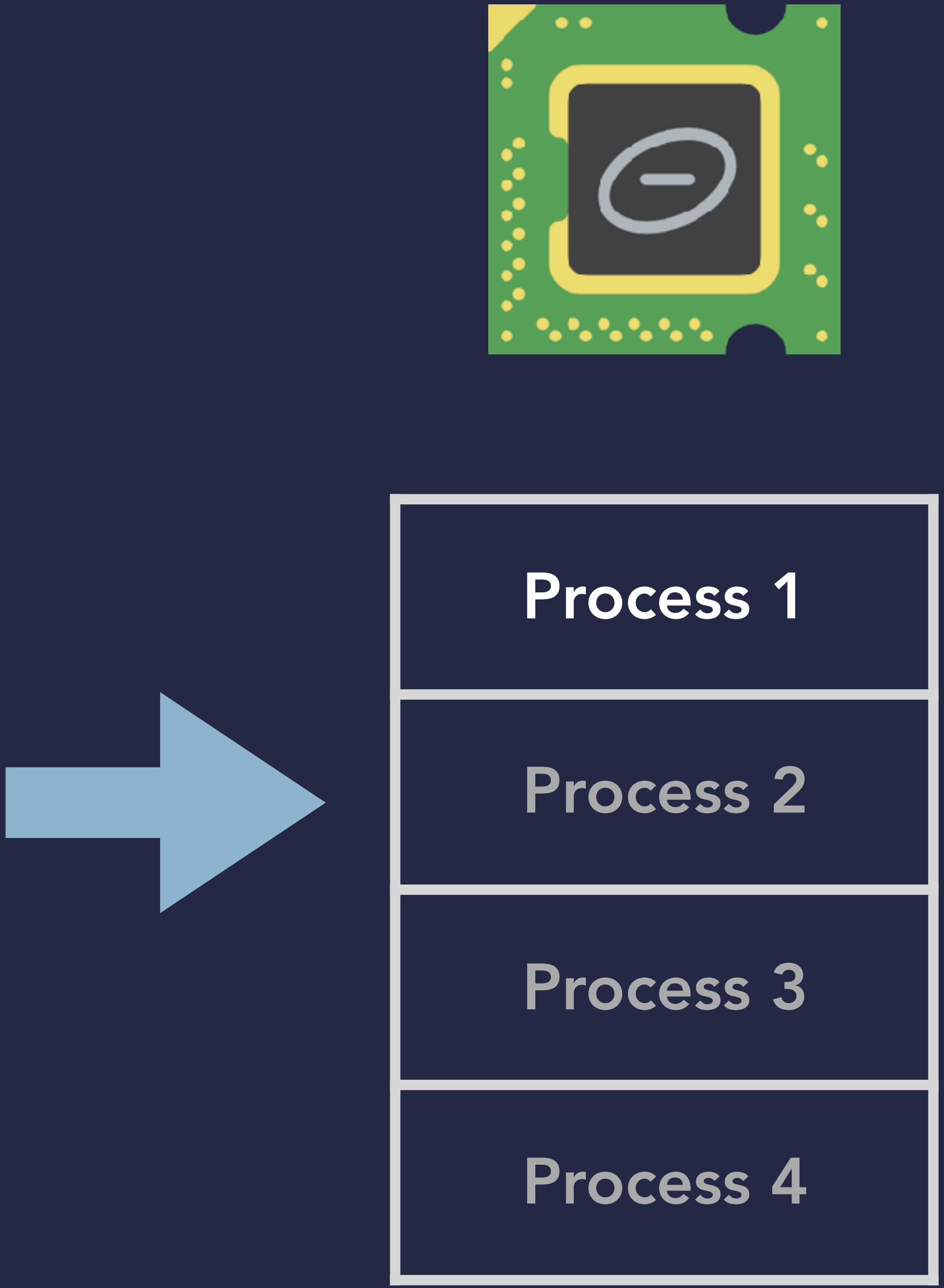
Poom Penghiran 5913873
CS2205 | Semester 1/2017

CPU Scheduling

What is CPU scheduling?



What is CPU scheduling?



Process 1

Process 2

Process 3

Process 4

CPU scheduling

FIFO

Preemptive (SFJ)

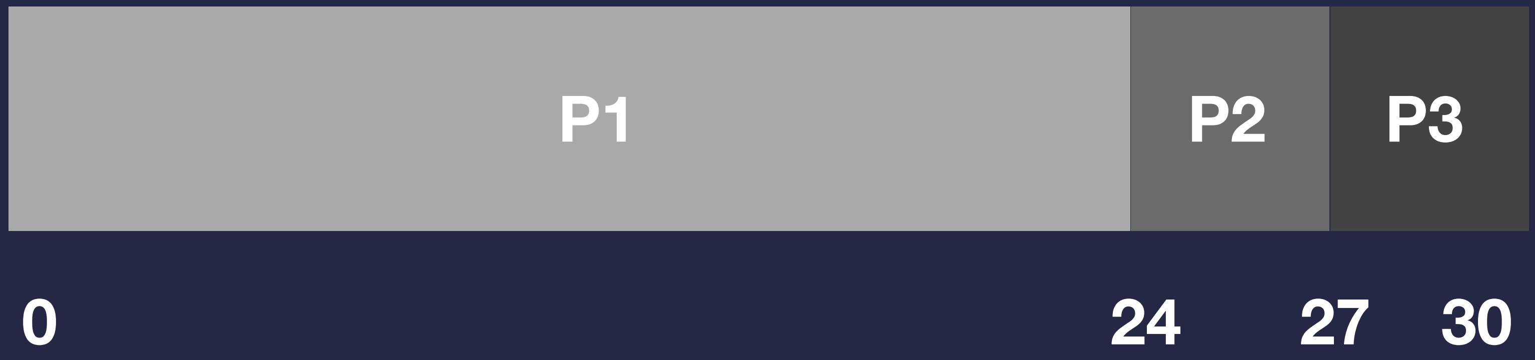
Priority

Round Robin

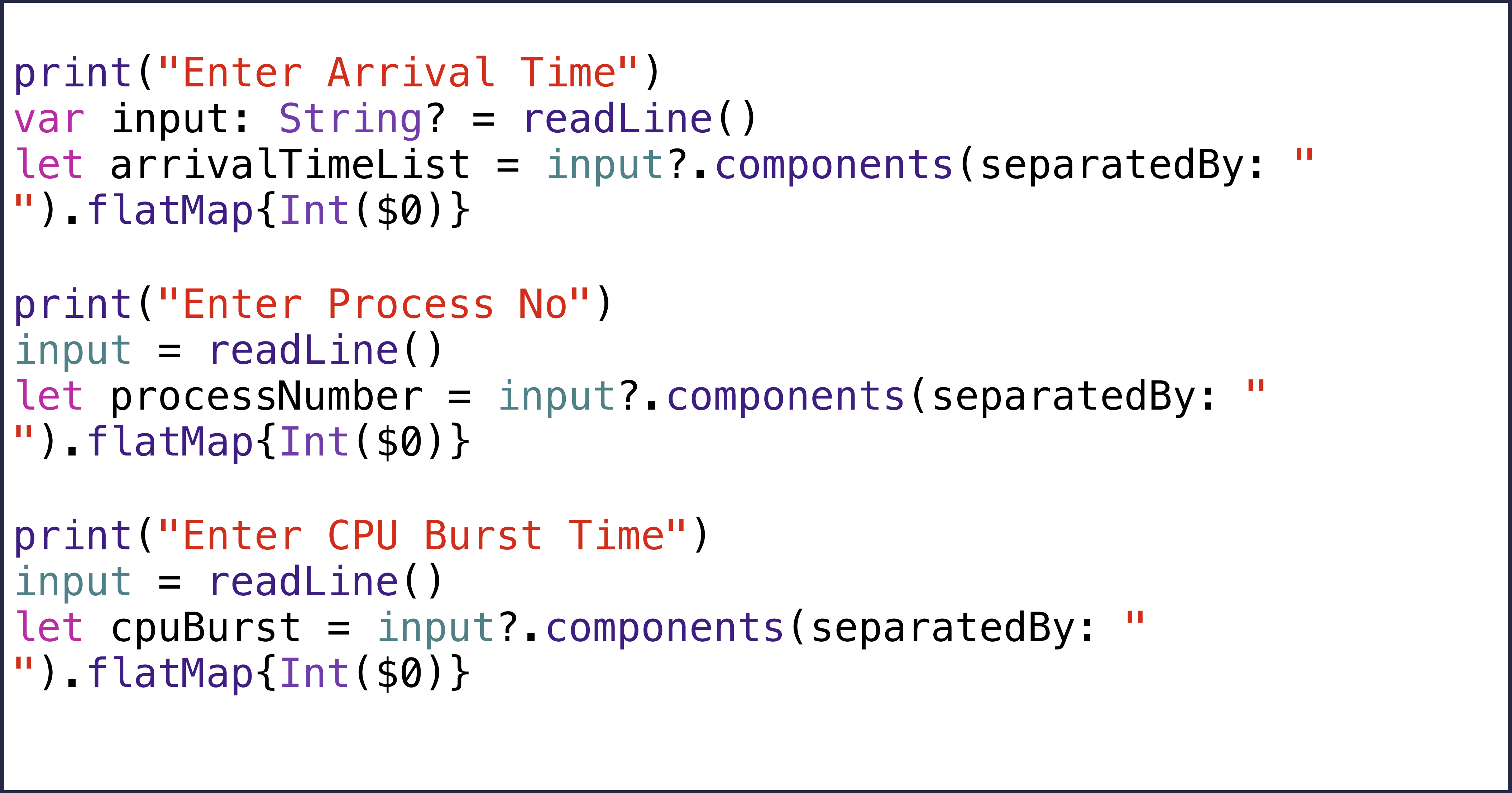
First In First Out

FIFO

Process	CPU Burst
P1	24
P2	3
P3	3



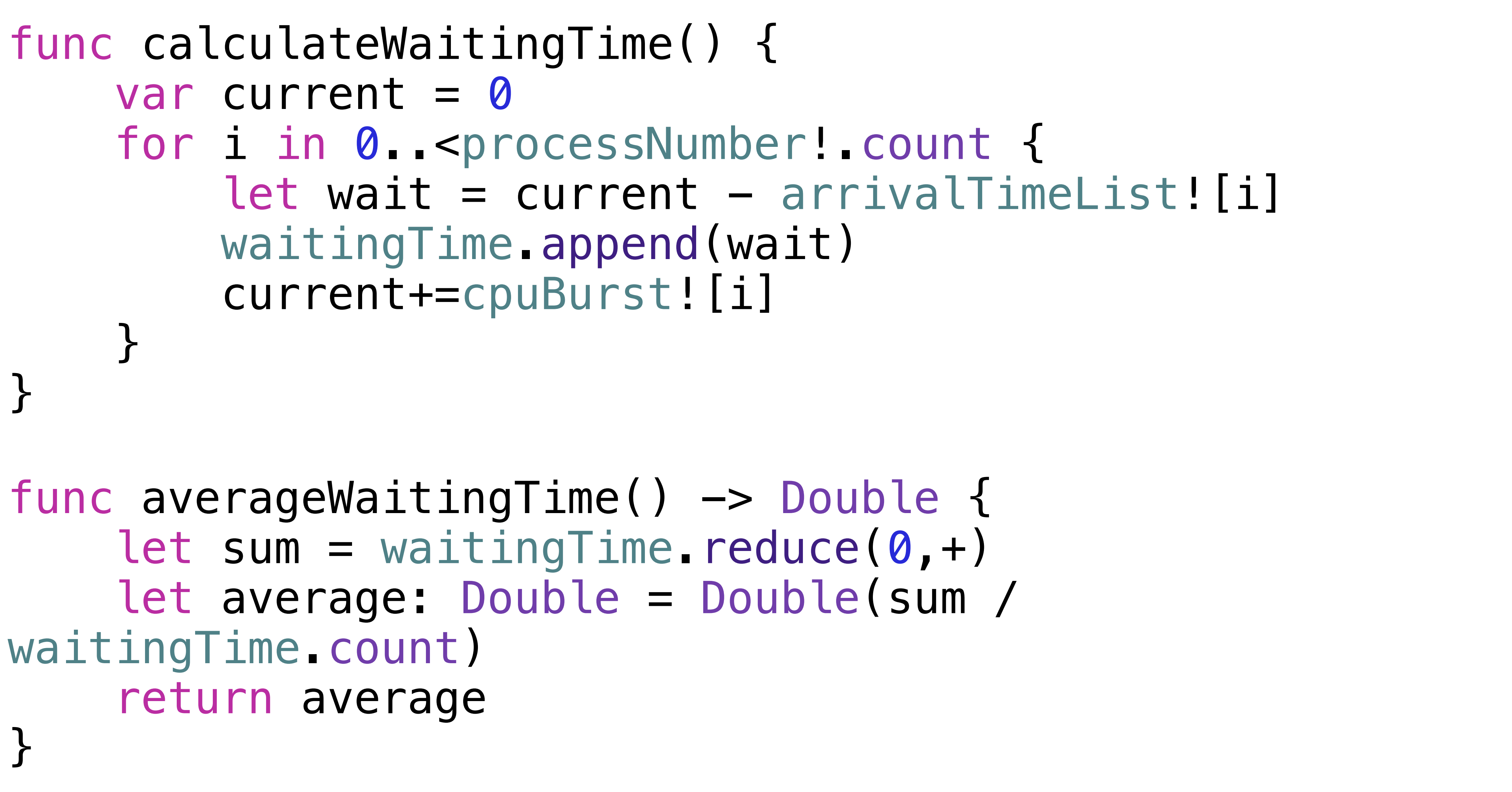
FIFO



```
print("Enter Arrival Time")
var input: String? = readLine()
let arrivalTimeList = input?.components(separatedBy: "
").flatMap{Int($0)}

print("Enter Process No")
input = readLine()
let processNumber = input?.components(separatedBy: "
").flatMap{Int($0)}

print("Enter CPU Burst Time")
input = readLine()
let cpuBurst = input?.components(separatedBy: "
").flatMap{Int($0)}
```

Input from the user

FIFO

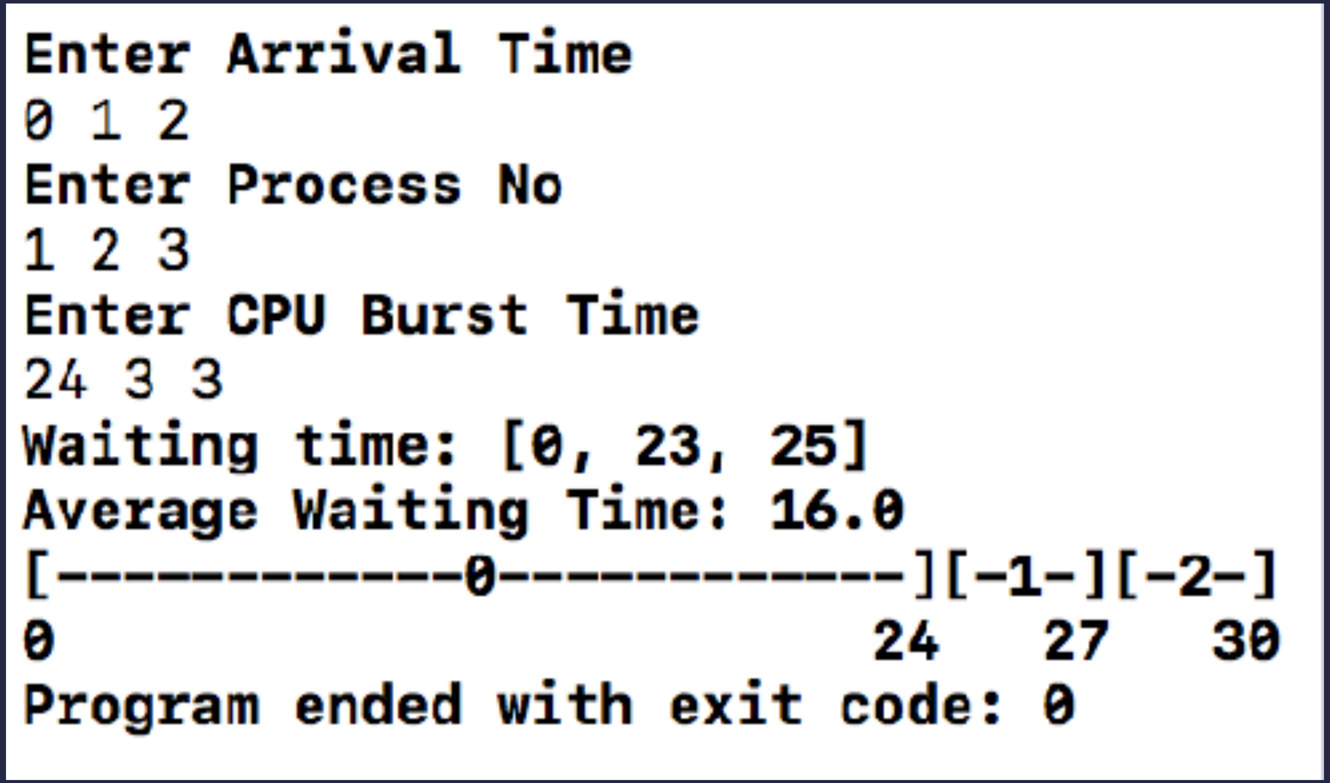


```
func calculateWaitingTime() {
    var current = 0
    for i in 0..<processNumber!.count {
        let wait = current - arrivalTimeList![i]
        waitingTime.append(wait)
        current+=cpuBurst![i]
    }
}

func averageWaitingTime() -> Double {
    let sum = waitingTime.reduce(0,+)
    let average: Double = Double(sum /
waitingTime.count)
    return average
}
```

Calculate waiting time and average waiting time

FIFO

Input and output of the program

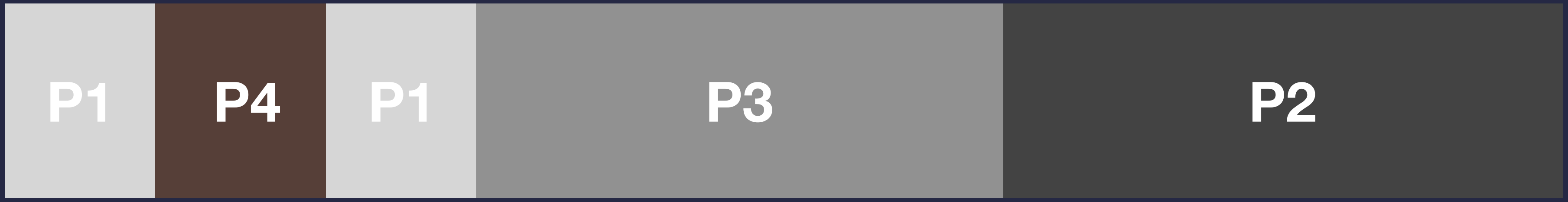


```
Enter Arrival Time
0 1 2
Enter Process No
1 2 3
Enter CPU Burst Time
24 3 3
Waiting time: [0, 23, 25]
Average Waiting Time: 16.0
[------------0------------][-1-][-2-]
0                        24   27   30
Program ended with exit code: 0
```

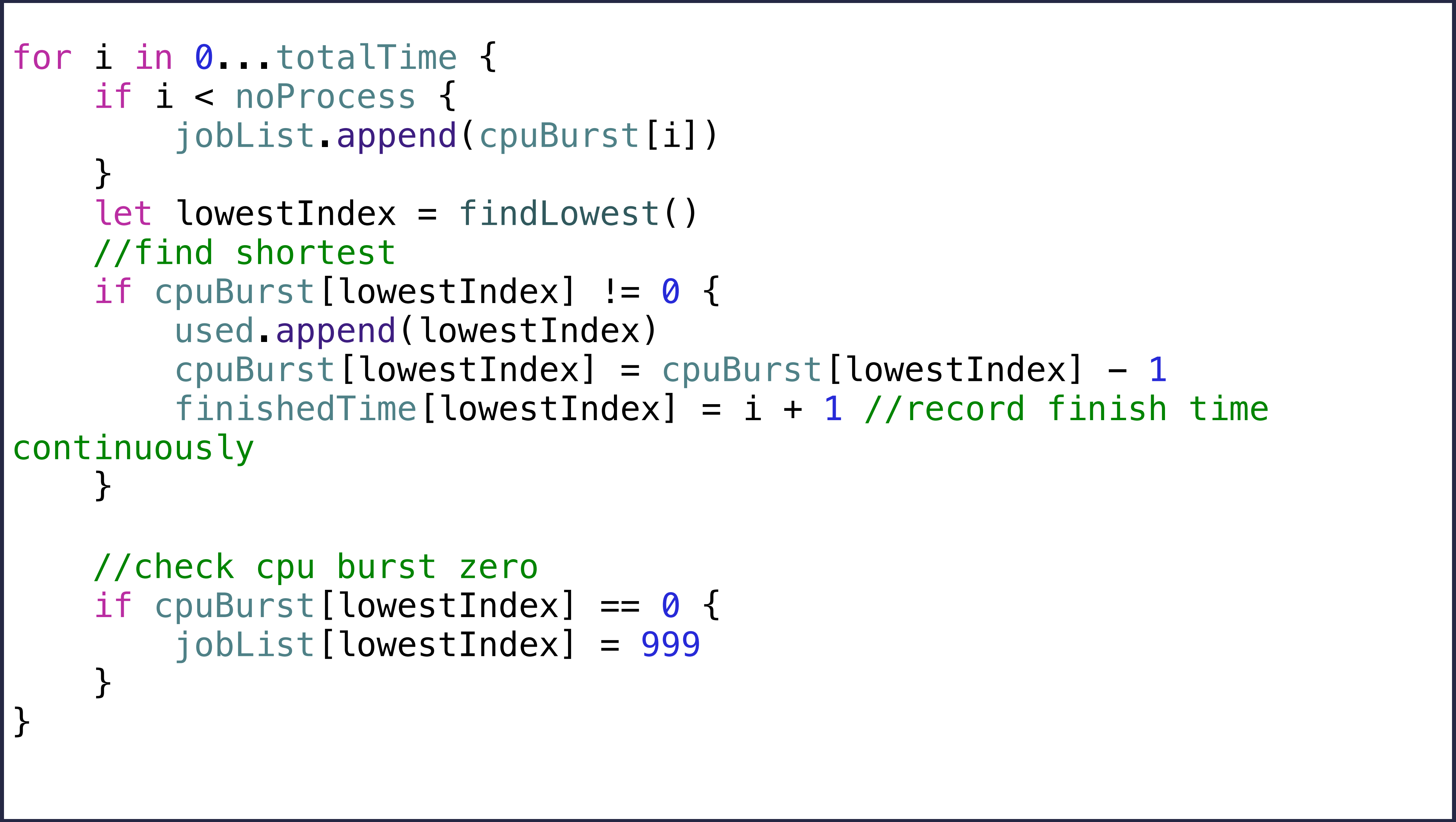
Shortest Job First

SFJ

Arrival Time	Process	CPU Burst
0	P1	6
1	P2	8
2	P3	7
3	P4	3



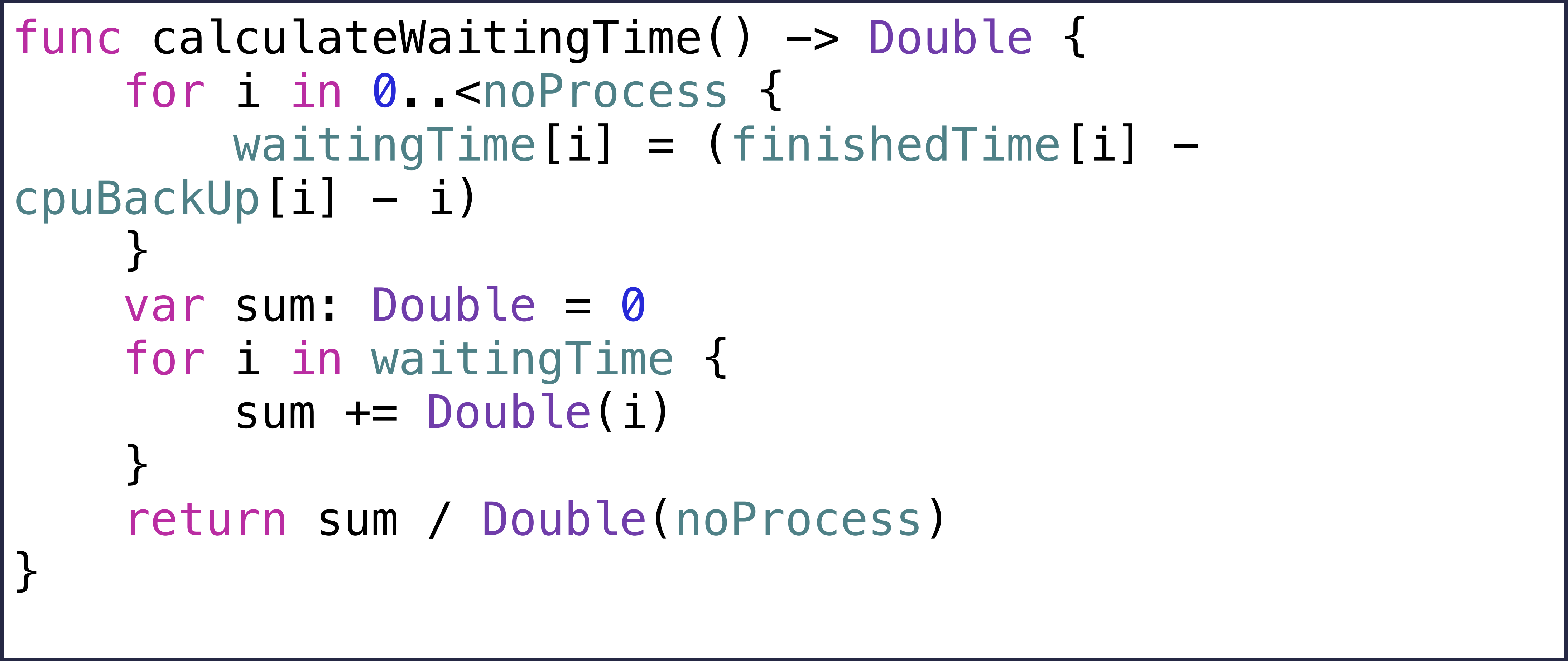
SFJ



```
for i in 0...totalTime {
    if i < noProcess {
        jobList.append(cpuBurst[i])
    }
    let lowestIndex = findLowest()
    //find shortest
    if cpuBurst[lowestIndex] != 0 {
        used.append(lowestIndex)
        cpuBurst[lowestIndex] = cpuBurst[lowestIndex] - 1
        finishedTime[lowestIndex] = i + 1 //record finish time
continuously
    }

    //check cpu burst zero
    if cpuBurst[lowestIndex] == 0 {
        jobList[lowestIndex] = 999
    }
}
```

Chooses the shortest job

SFJ

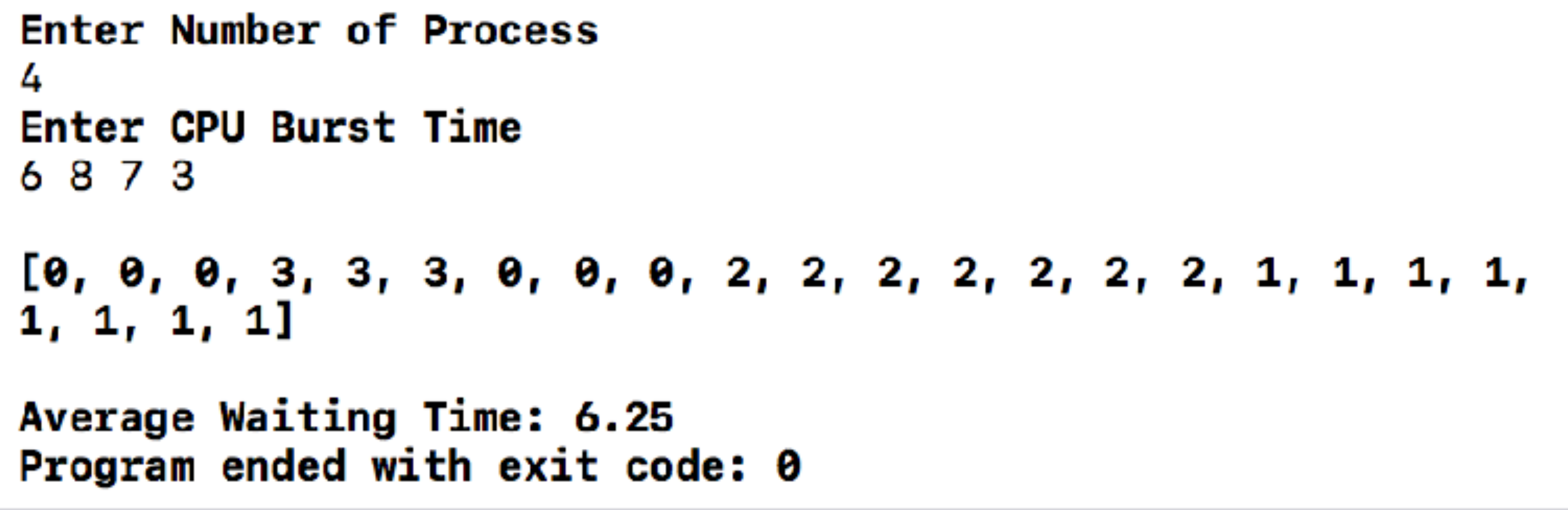


```
func calculateWaitingTime() -> Double {
    for i in 0..<noProcess {
        waitingTime[i] = (finishedTime[i] -
cpuBackUp[i] - i)
    }
    var sum: Double = 0
    for i in waitingTime {
        sum += Double(i)
    }
    return sum / Double(noProcess)
}
```

Calculating average waiting time

SFJ

Input and output of the program

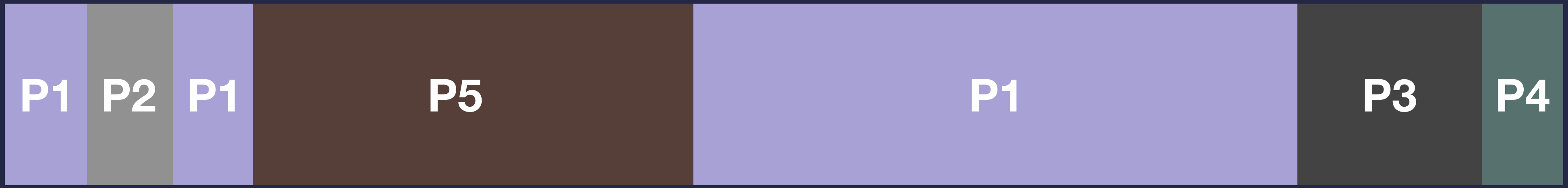


```
Enter Number of Process
4
Enter CPU Burst Time
6 8 7 3

[0, 0, 0, 3, 3, 3, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1,
1, 1, 1, 1]

Average Waiting Time: 6.25
Program ended with exit code: 0
```

Priority Scheduling

Priority Scheduling

Arrival Time	Process	CPU Burst	Prioriy
0	P1	10	3
1	P2	1	1
2	P3	2	4
3	P4	1	5
4	P5	5	2

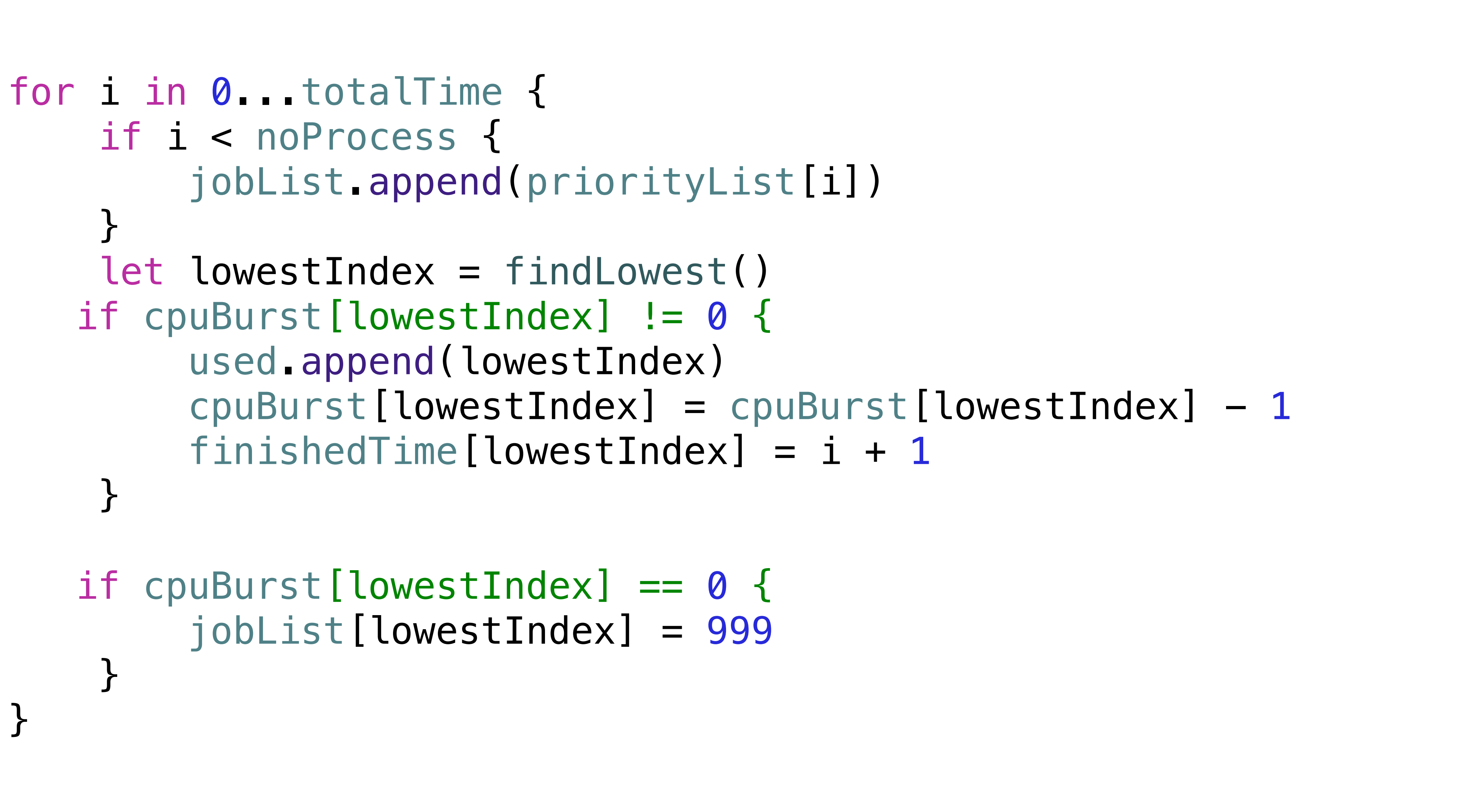


Priority Scheduling

```
print("Enter CPU Priority")
input = readLine()
var priorityList = input!.components(separatedBy: "
").flatMap{Int($0)}
```

Input priority from user

Priority Scheduling

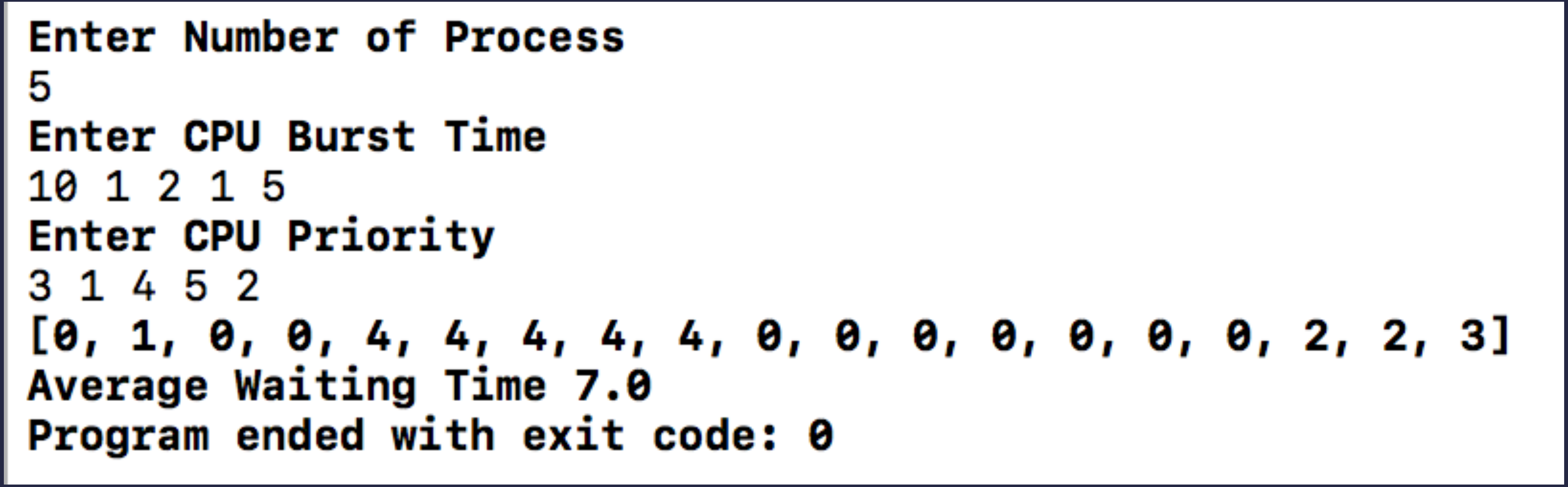


```
for i in 0...totalTime {
    if i < noProcess {
        jobList.append(priorityList[i])
    }
    let lowestIndex = findLowest()
   if cpuBurst[lowestIndex] != 0 {
        used.append(lowestIndex)
        cpuBurst[lowestIndex] = cpuBurst[lowestIndex] - 1
        finishedTime[lowestIndex] = i + 1
    }

   if cpuBurst[lowestIndex] == 0 {
        jobList[lowestIndex] = 999
    }
}
```

Chooses the highest priority job

Priority Scheduling

Input and output of the program

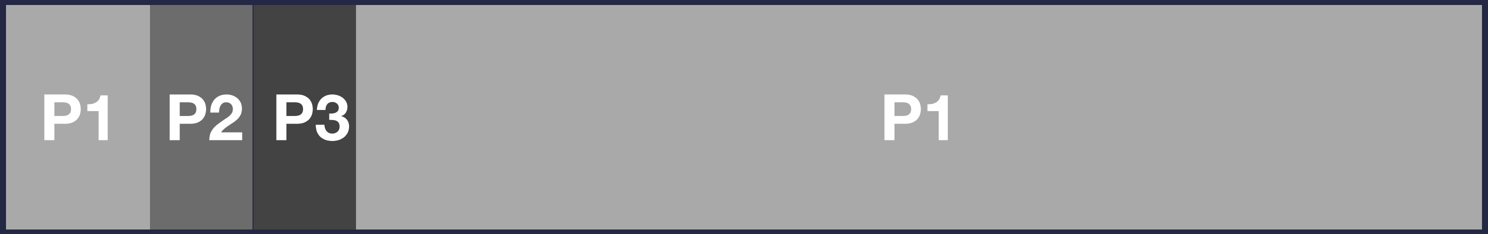


```
Enter Number of Process
5
Enter CPU Burst Time
10 1 2 1 5
Enter CPU Priority
3 1 4 5 2
[0, 1, 0, 0, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 2, 2, 3]
Average Waiting Time 7.0
Program ended with exit code: 0
```

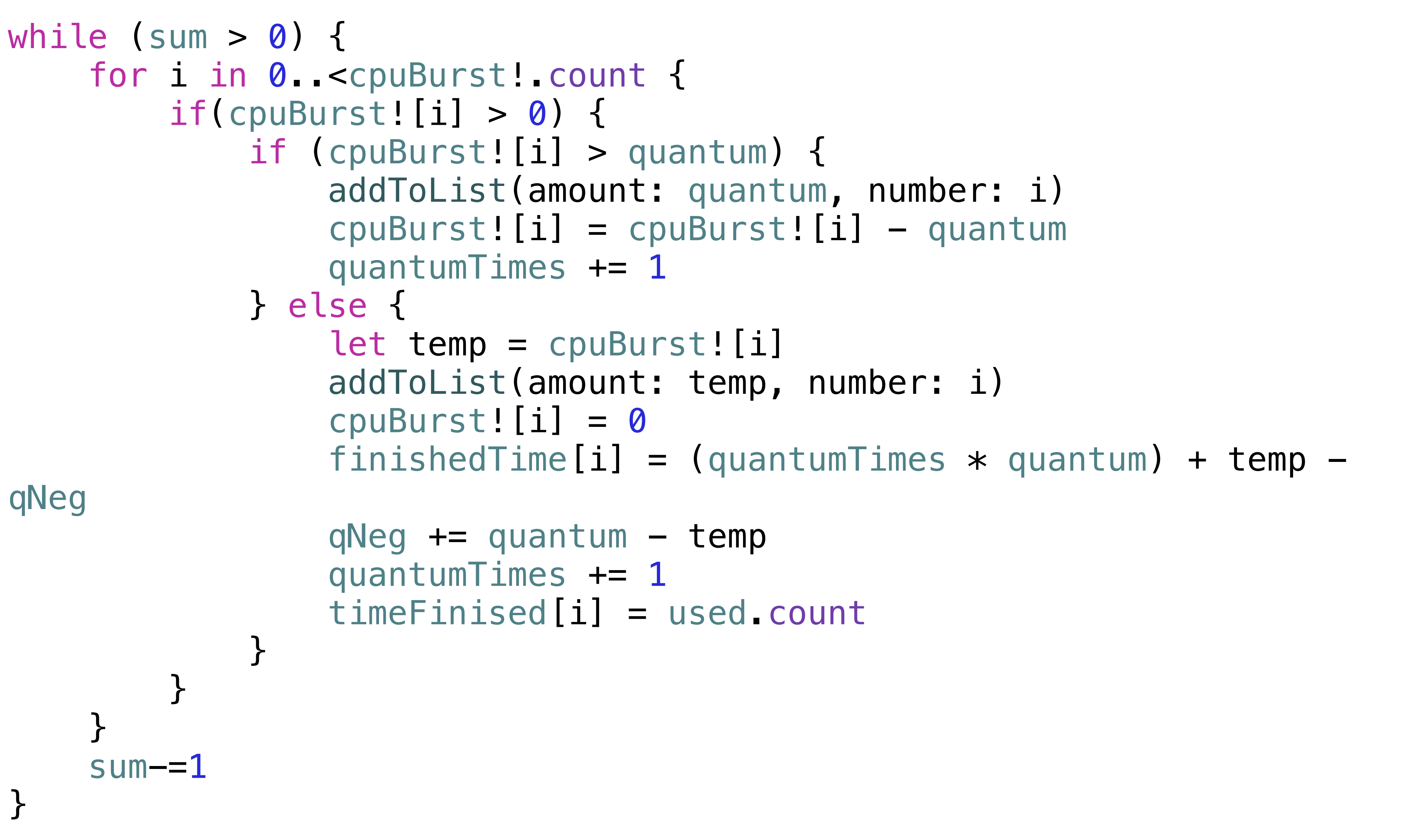
Round-Robin

Round-Robin

Process	CPU Burst
P1	24
P2	3
P3	3



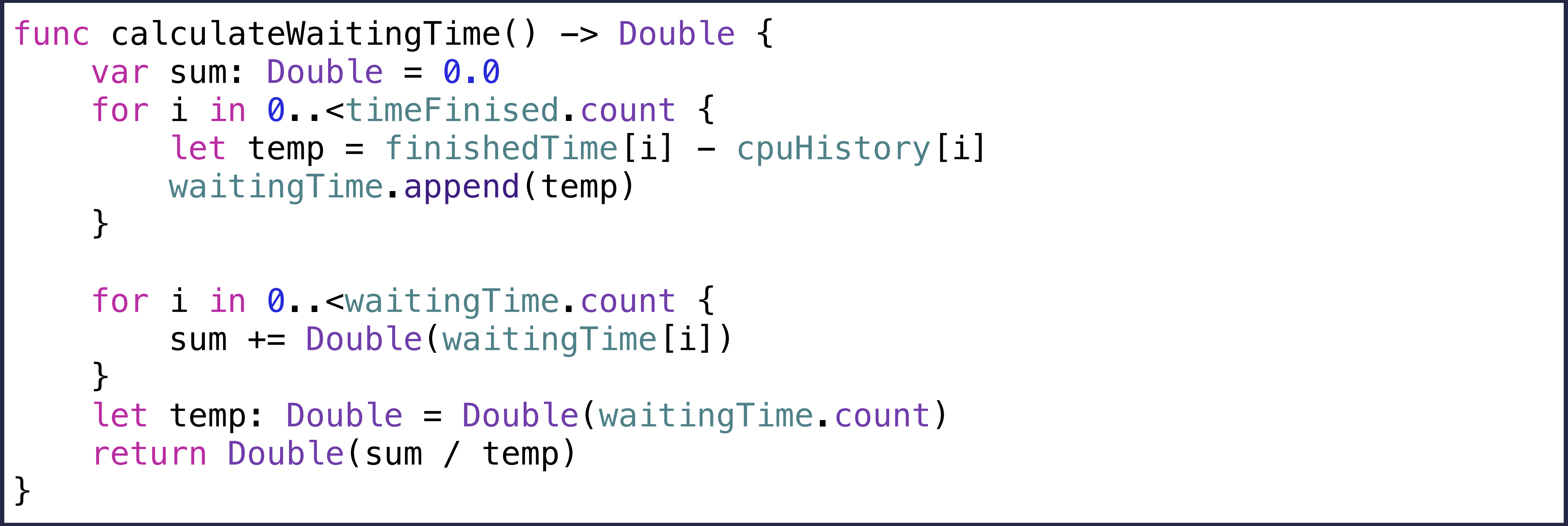
Round-Robin



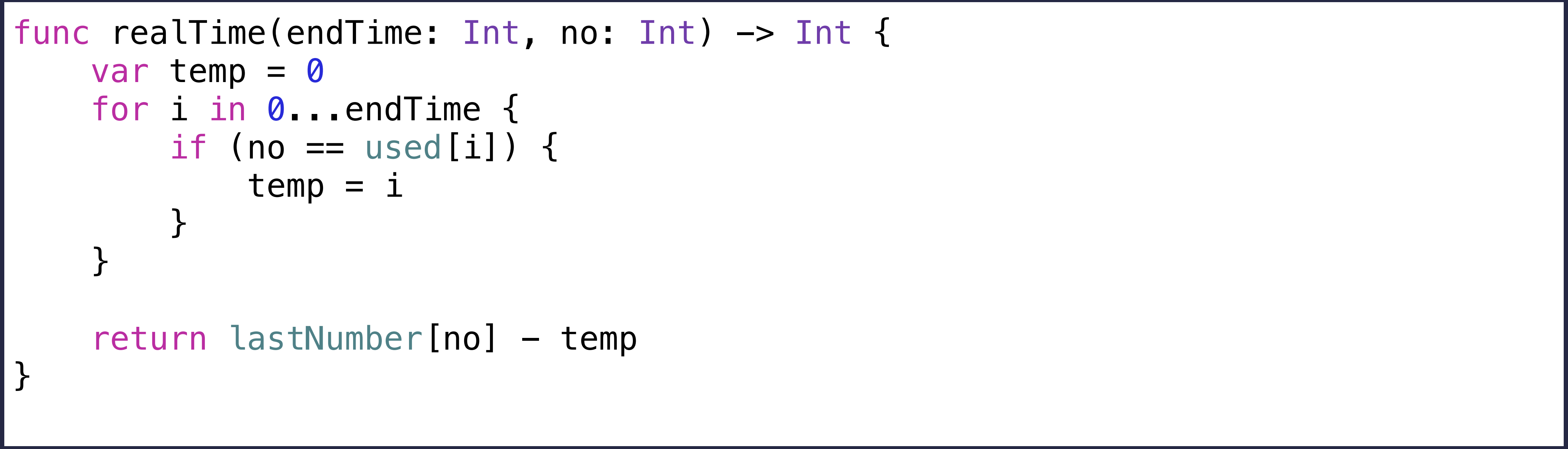
```
while (sum > 0) {
    for i in 0..<cpuBurst!.count {
        if(cpuBurst![i] > 0) {
            if (cpuBurst![i] > quantum) {
                addToList(amount: quantum, number: i)
                cpuBurst![i] = cpuBurst![i] - quantum
                quantumTimes += 1
            } else {
                let temp = cpuBurst![i]
                addToList(amount: temp, number: i)
                cpuBurst![i] = 0
                finishedTime[i] = (quantumTimes * quantum) + temp -
qNeg
                qNeg += quantum - temp
                quantumTimes += 1
                timeFinised[i] = used.count
            }
        }
    }
    sum-=1
}
```

Operating a circular job

Round-Robin



```swift
func calculateWaitingTime() -> Double {
    var sum: Double = 0.0
    for i in 0..<timeFinised.count {
        let temp = finishedTime[i] - cpuHistory[i]
        waitingTime.append(temp)
    }

    for i in 0..<waitingTime.count {
        sum += Double(waitingTime[i])
    }
    let temp: Double = Double(waitingTime.count)
    return Double(sum / temp)
}
```

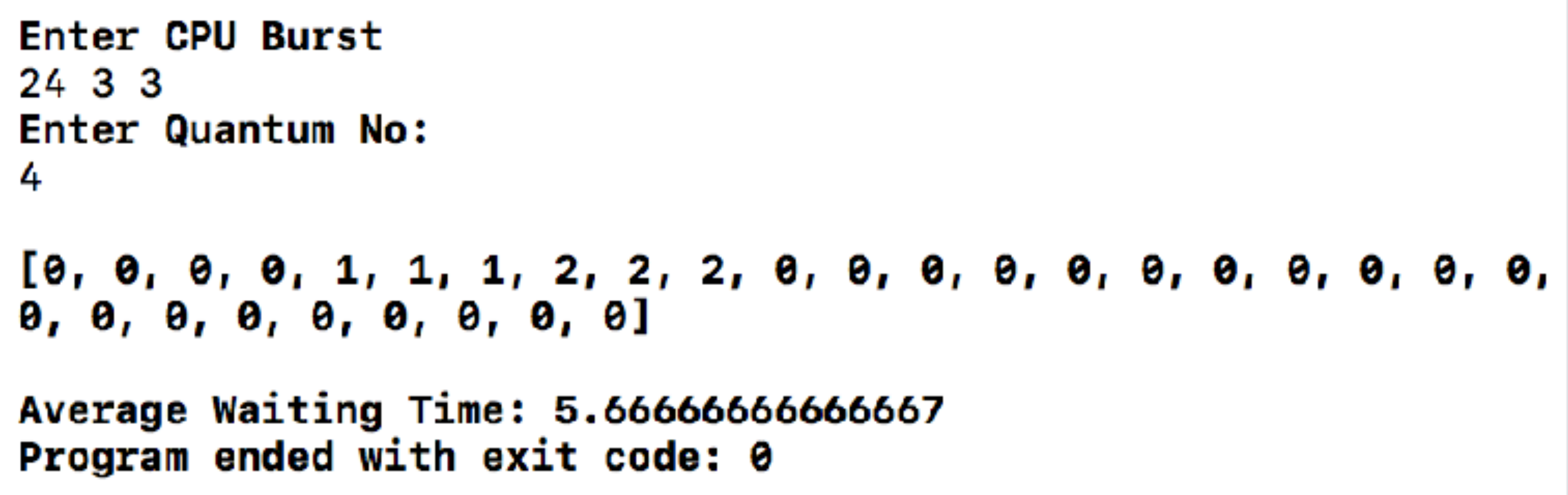


```swift
func realTime(endTime: Int, no: Int) -> Int {
    var temp = 0
    for i in 0...endTime {
        if (no == used[i]) {
            temp = i
        }
    }

    return lastNumber[no] - temp
}
```

Calculating average waiting time

Round-Robin

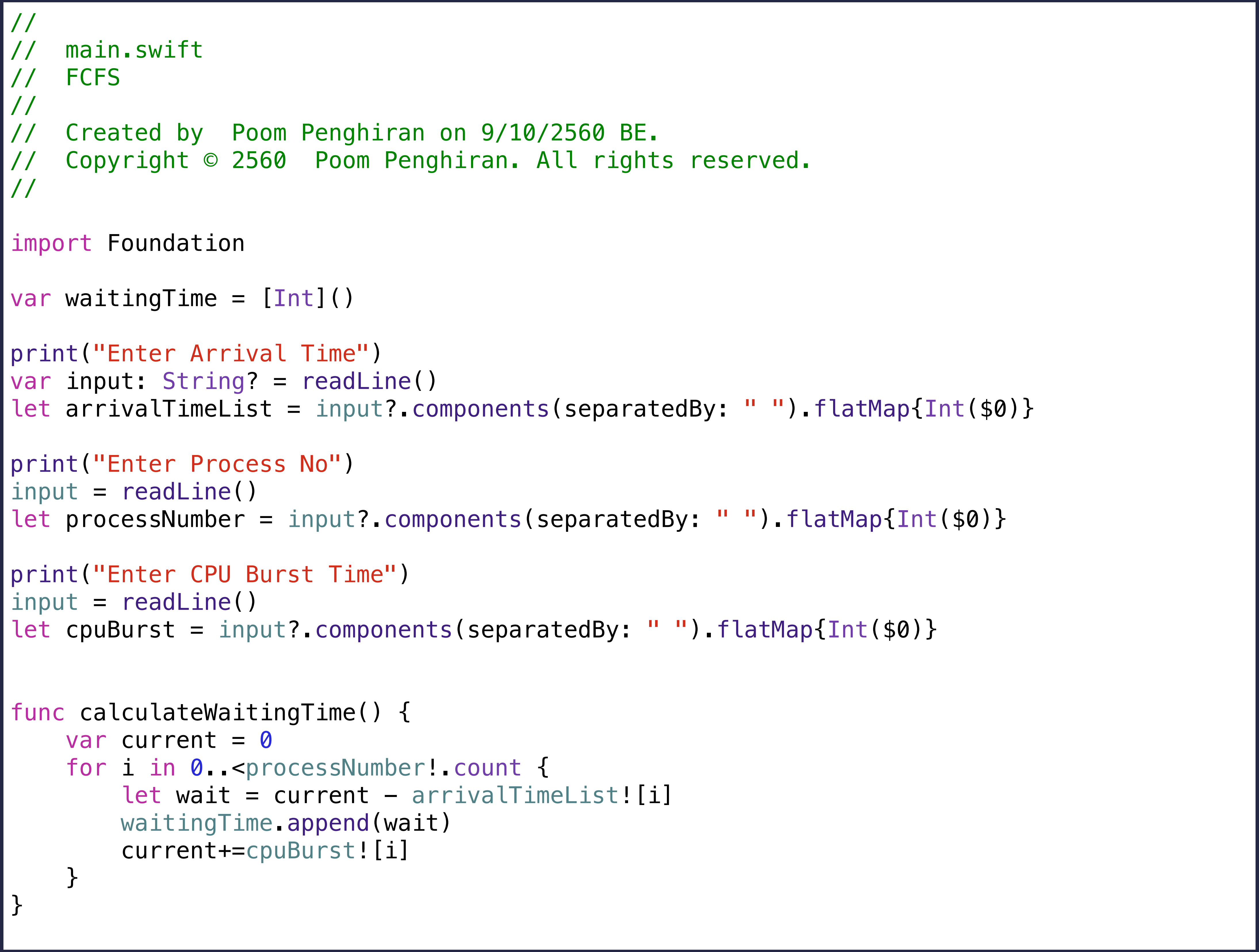
Input and output of the program



```
Enter CPU Burst
24 3 3
Enter Quantum No:
4

[0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0]

Average Waiting Time: 5.666666666666667
Program ended with exit code: 0
```

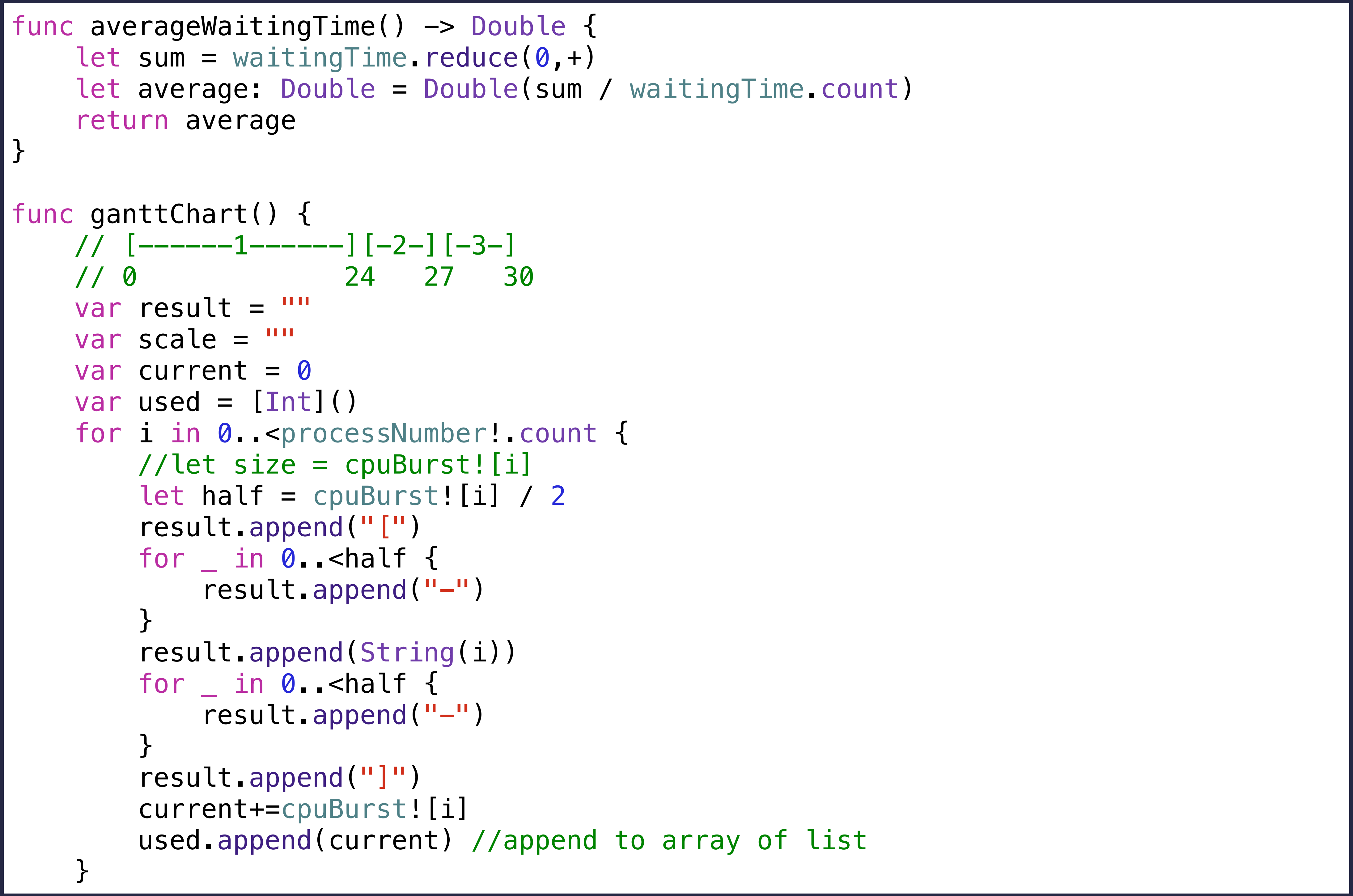
Q&A

Reference - FIFO



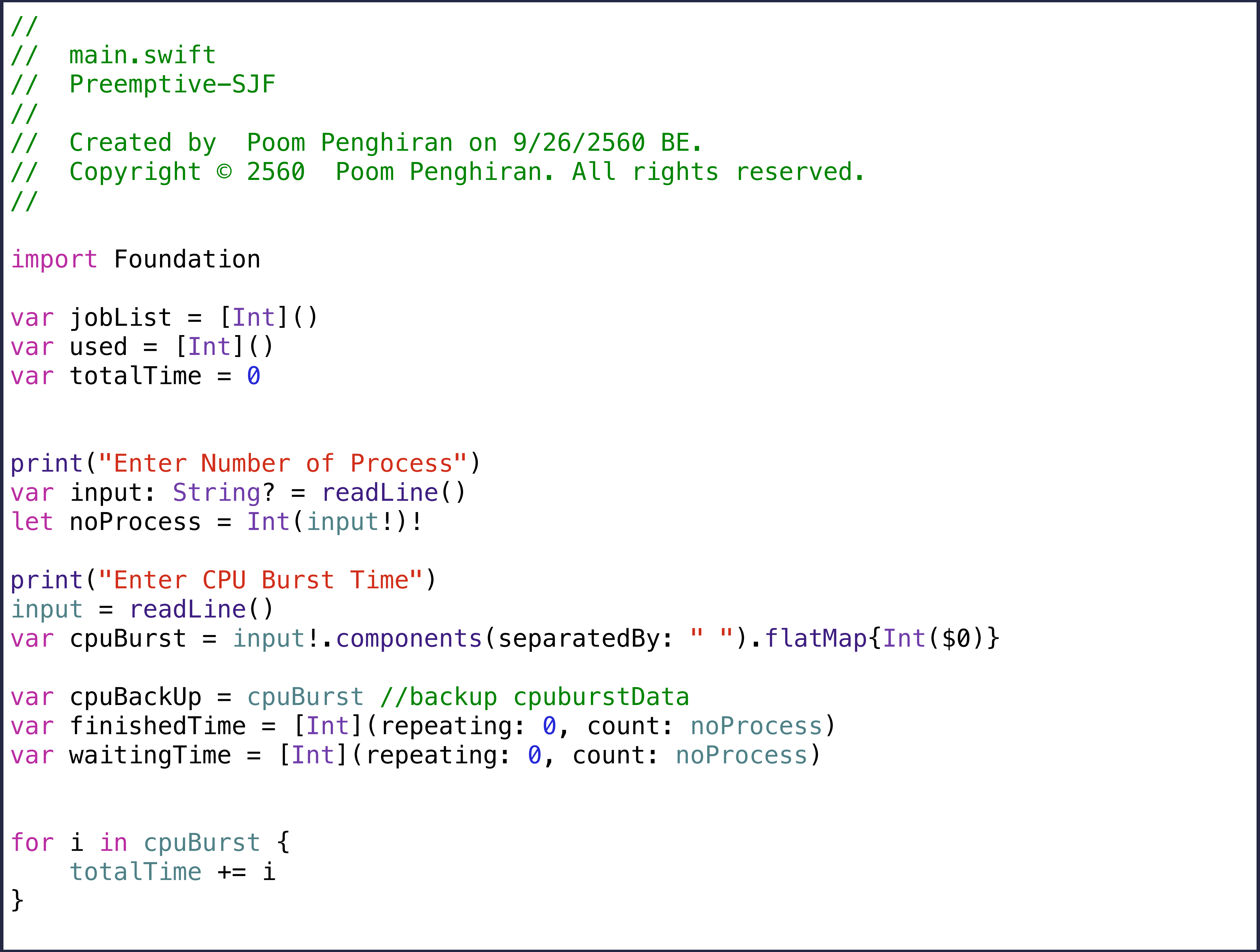
```swift
//
//  main.swift
//  FCFS
//
//  Created by  Poom Penghiran on 9/10/2560 BE.
//  Copyright © 2560  Poom Penghiran. All rights reserved.
//

import Foundation

var waitingTime = [Int]()

print("Enter Arrival Time")
var input: String? = readLine()
let arrivalTimeList = input?.components(separatedBy: " ").flatMap{Int($0)}

print("Enter Process No")
input = readLine()
let processNumber = input?.components(separatedBy: " ").flatMap{Int($0)}

print("Enter CPU Burst Time")
input = readLine()
let cpuBurst = input?.components(separatedBy: " ").flatMap{Int($0)}


func calculateWaitingTime() {
    var current = 0
    for i in 0..<processNumber!.count {
        let wait = current - arrivalTimeList![i]
        waitingTime.append(wait)
        current+=cpuBurst![i]
    }
}
```

Reference - FIFO



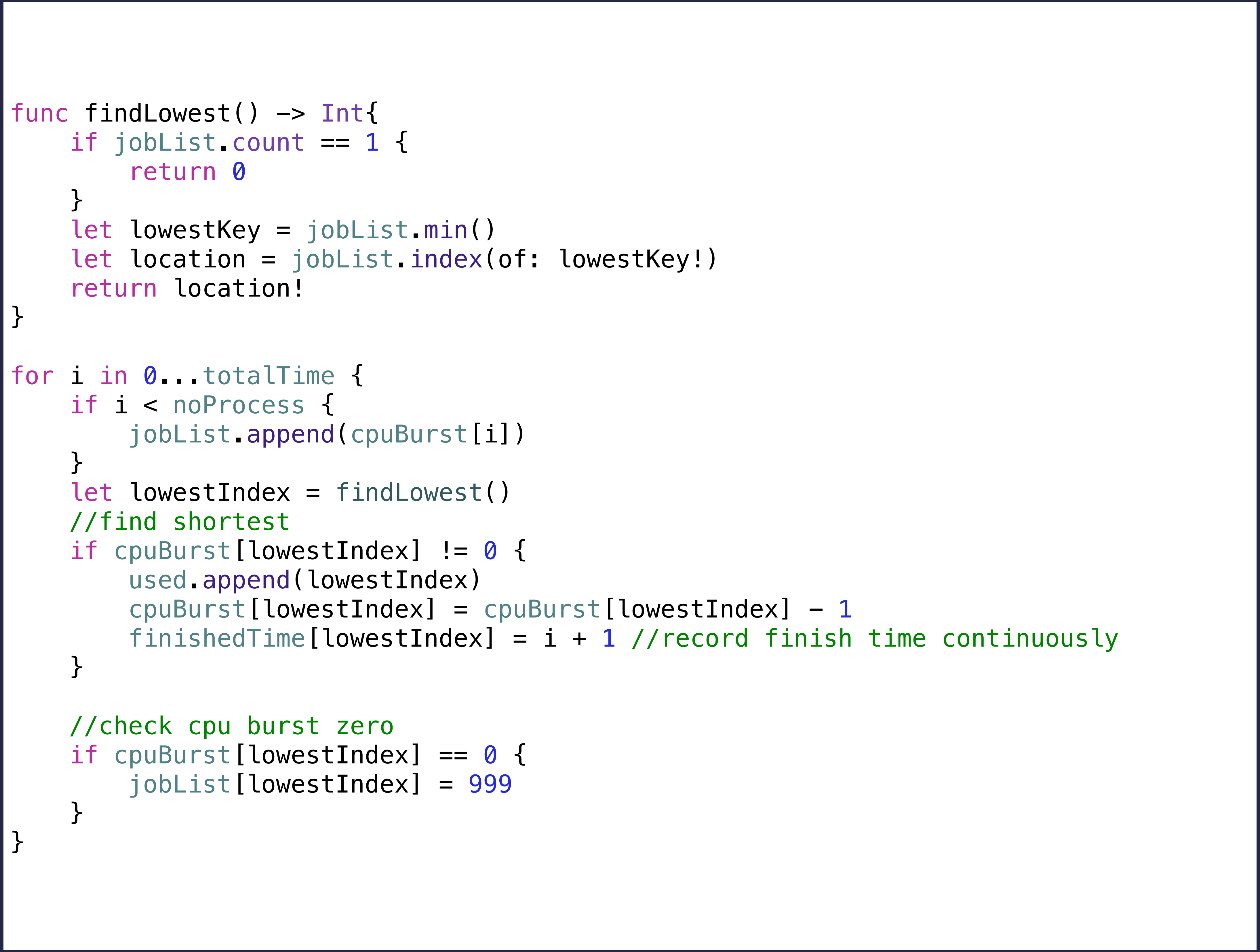
```
func averageWaitingTime() -> Double {
    let sum = waitingTime.reduce(0,+)
    let average: Double = Double(sum / waitingTime.count)
    return average
}

func ganttChart() {
    // [------1------][-2-][-3-]
    // 0             24   27   30
    var result = ""
    var scale = ""
    var current = 0
    var used = [Int]()
    for i in 0..<processNumber!.count {
        //let size = cpuBurst![i]
        let half = cpuBurst![i] / 2
        result.append("[")
        for _ in 0..<half {
            result.append("-")
        }
        result.append(String(i))
        for _ in 0..<half {
            result.append("-")
        }
        result.append("]")
        current+=cpuBurst![i]
        used.append(current) //append to array of list
    }
```

Reference - SJF



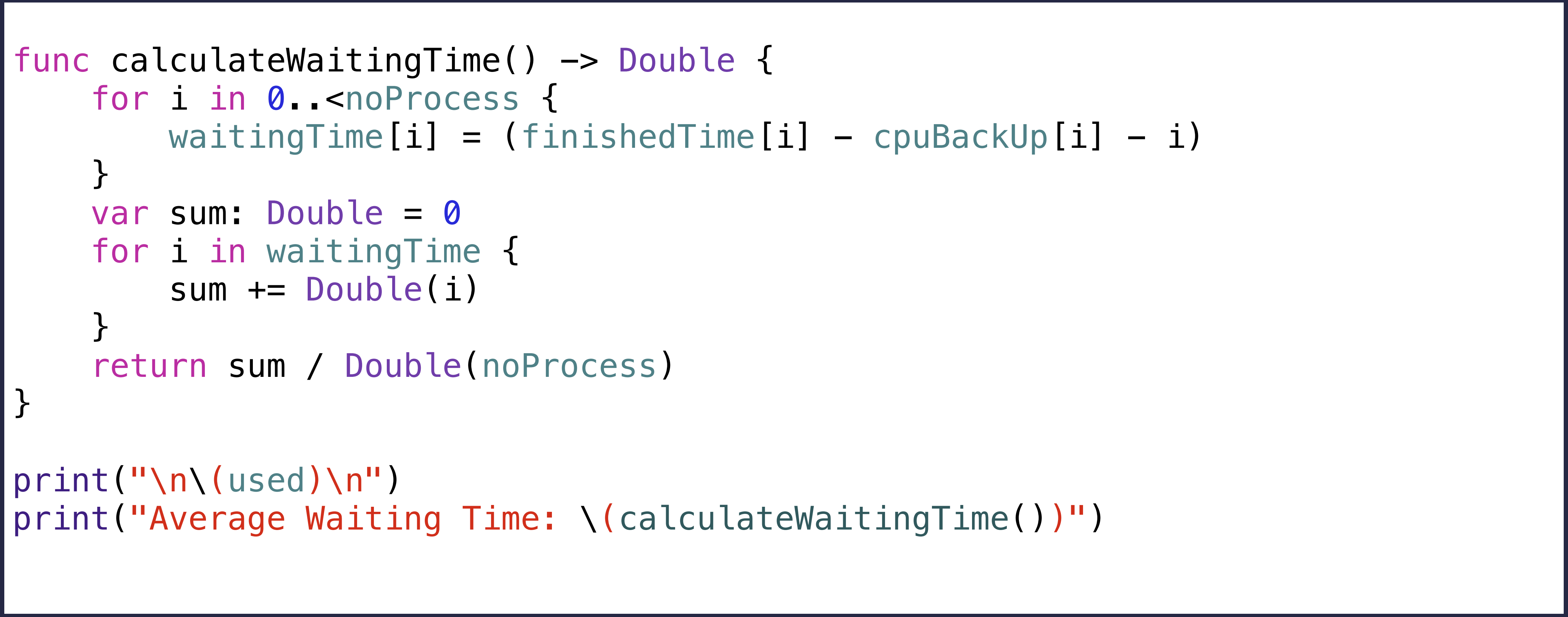
```swift
//
//  main.swift
//  Preemptive-SJF
//
//  Created by  Poom Penghiran on 9/26/2560 BE.
//  Copyright © 2560  Poom Penghiran. All rights reserved.
//

import Foundation

var jobList = [Int]()
var used = [Int]()
var totalTime = 0


print("Enter Number of Process")
var input: String? = readLine()
let noProcess = Int(input!)!

print("Enter CPU Burst Time")
input = readLine()
var cpuBurst = input!.components(separatedBy: " ").flatMap{Int($0)}

var cpuBackUp = cpuBurst //backup cpuburstData
var finishedTime = [Int](repeating: 0, count: noProcess)
var waitingTime = [Int](repeating: 0, count: noProcess)


for i in cpuBurst {
    totalTime += i
}
```

Reference - SJF



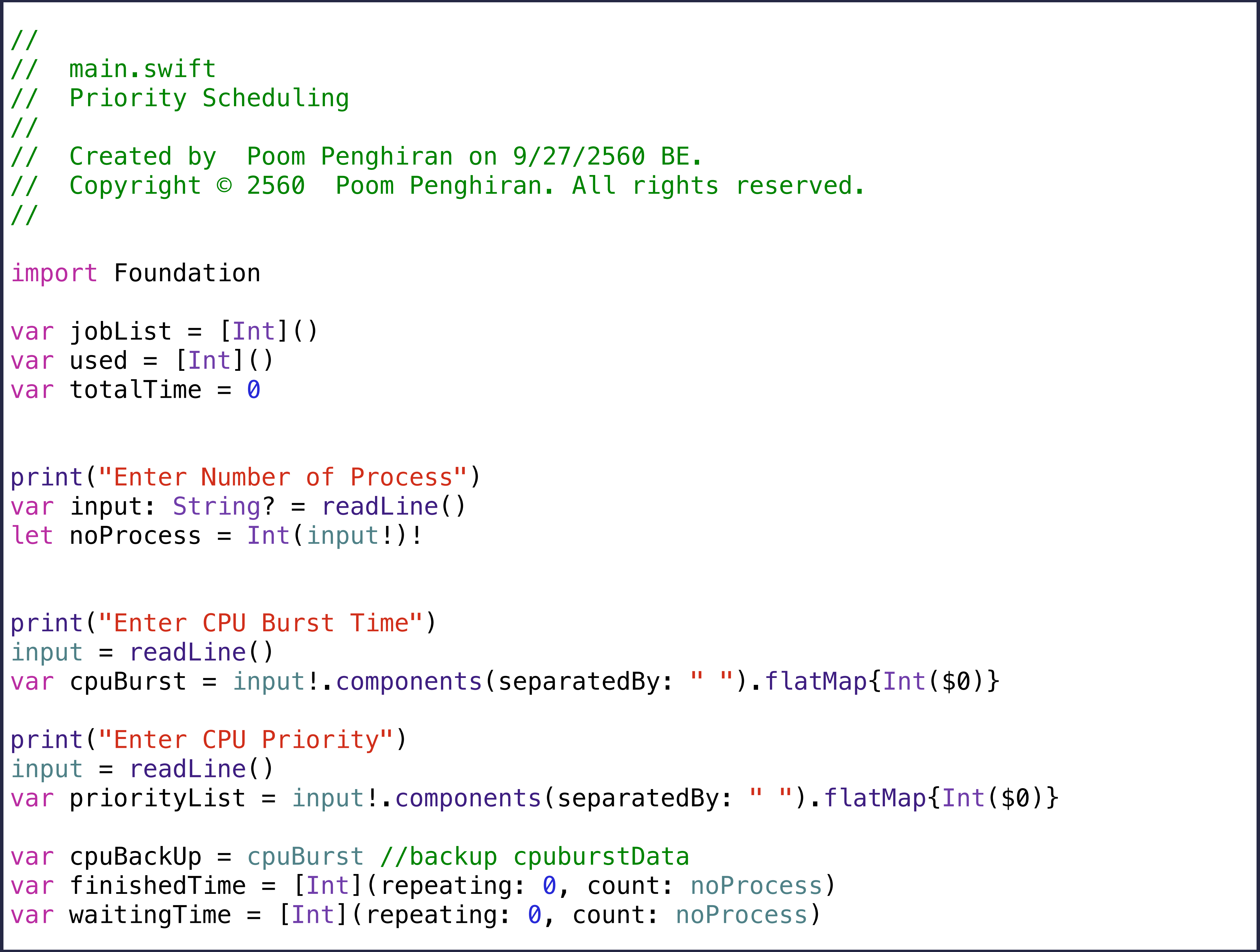
```
func findLowest() -> Int{
    if jobList.count == 1 {
        return 0
    }
    let lowestKey = jobList.min()
    let location = jobList.index(of: lowestKey!)
    return location!
}

for i in 0...totalTime {
    if i < noProcess {
        jobList.append(cpuBurst[i])
    }
    let lowestIndex = findLowest()
    //find shortest
    if cpuBurst[lowestIndex] != 0 {
        used.append(lowestIndex)
        cpuBurst[lowestIndex] = cpuBurst[lowestIndex] - 1
        finishedTime[lowestIndex] = i + 1 //record finish time continuously
    }

    //check cpu burst zero
    if cpuBurst[lowestIndex] == 0 {
        jobList[lowestIndex] = 999
    }
}
```

Reference - SJF



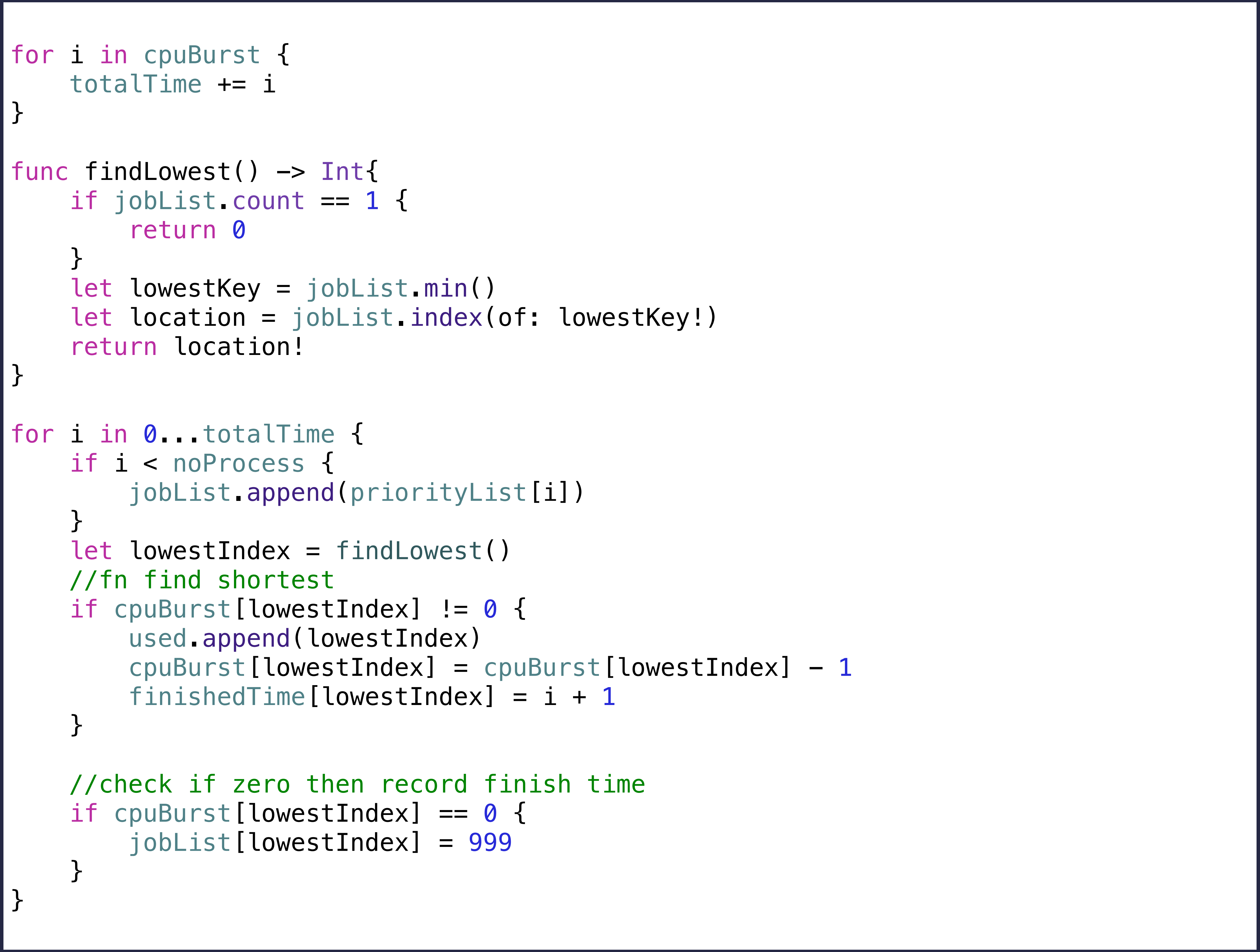
```
func calculateWaitingTime() -> Double {
    for i in 0..<noProcess {
        waitingTime[i] = (finishedTime[i] - cpuBackUp[i] - i)
    }
    var sum: Double = 0
    for i in waitingTime {
        sum += Double(i)
    }
    return sum / Double(noProcess)
}

print("\n\(used)\n")
print("Average Waiting Time: \(calculateWaitingTime())")
```

Reference - Priority Scheduling



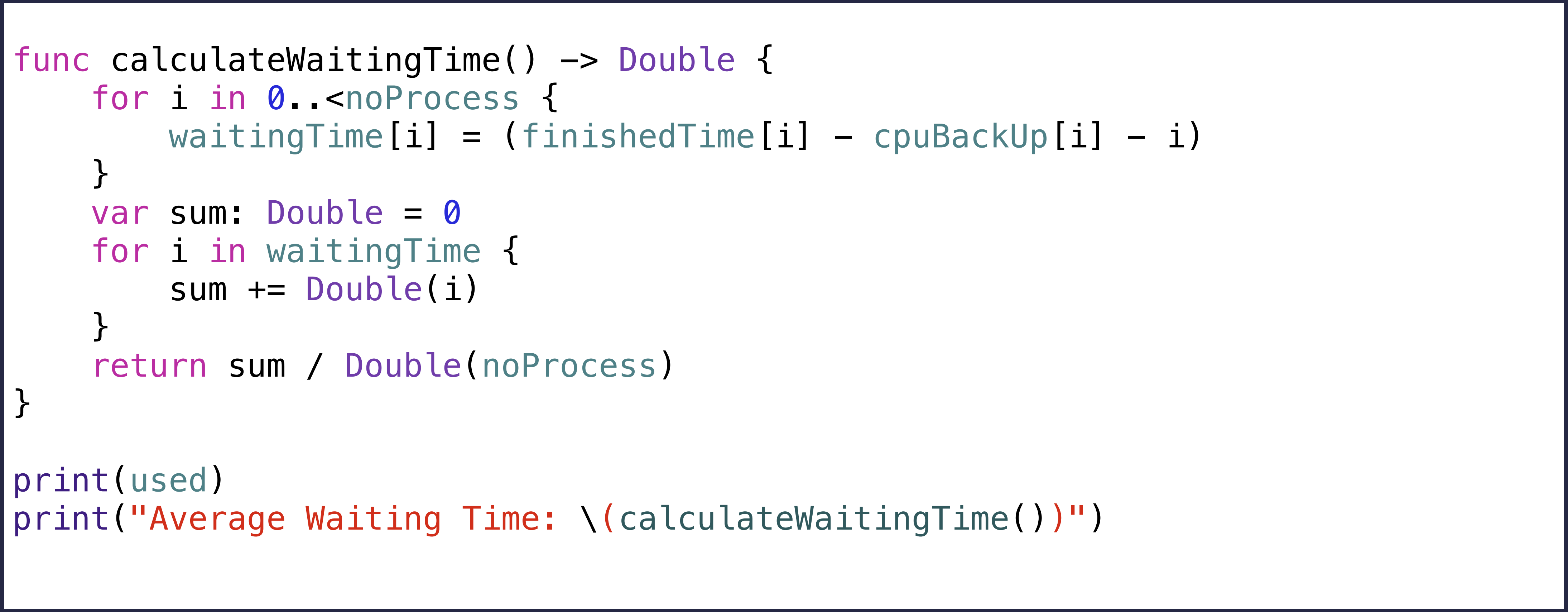
```swift
//
//  main.swift
//  Priority Scheduling
//
//  Created by  Poom Penghiran on 9/27/2560 BE.
//  Copyright © 2560  Poom Penghiran. All rights reserved.
//

import Foundation

var jobList = [Int]()
var used = [Int]()
var totalTime = 0


print("Enter Number of Process")
var input: String? = readLine()
let noProcess = Int(input!)!


print("Enter CPU Burst Time")
input = readLine()
var cpuBurst = input!.components(separatedBy: " ").flatMap{Int($0)}

print("Enter CPU Priority")
input = readLine()
var priorityList = input!.components(separatedBy: " ").flatMap{Int($0)}

var cpuBackUp = cpuBurst //backup cpuburstData
var finishedTime = [Int](repeating: 0, count: noProcess)
var waitingTime = [Int](repeating: 0, count: noProcess)
```

Reference - Priority Scheduling



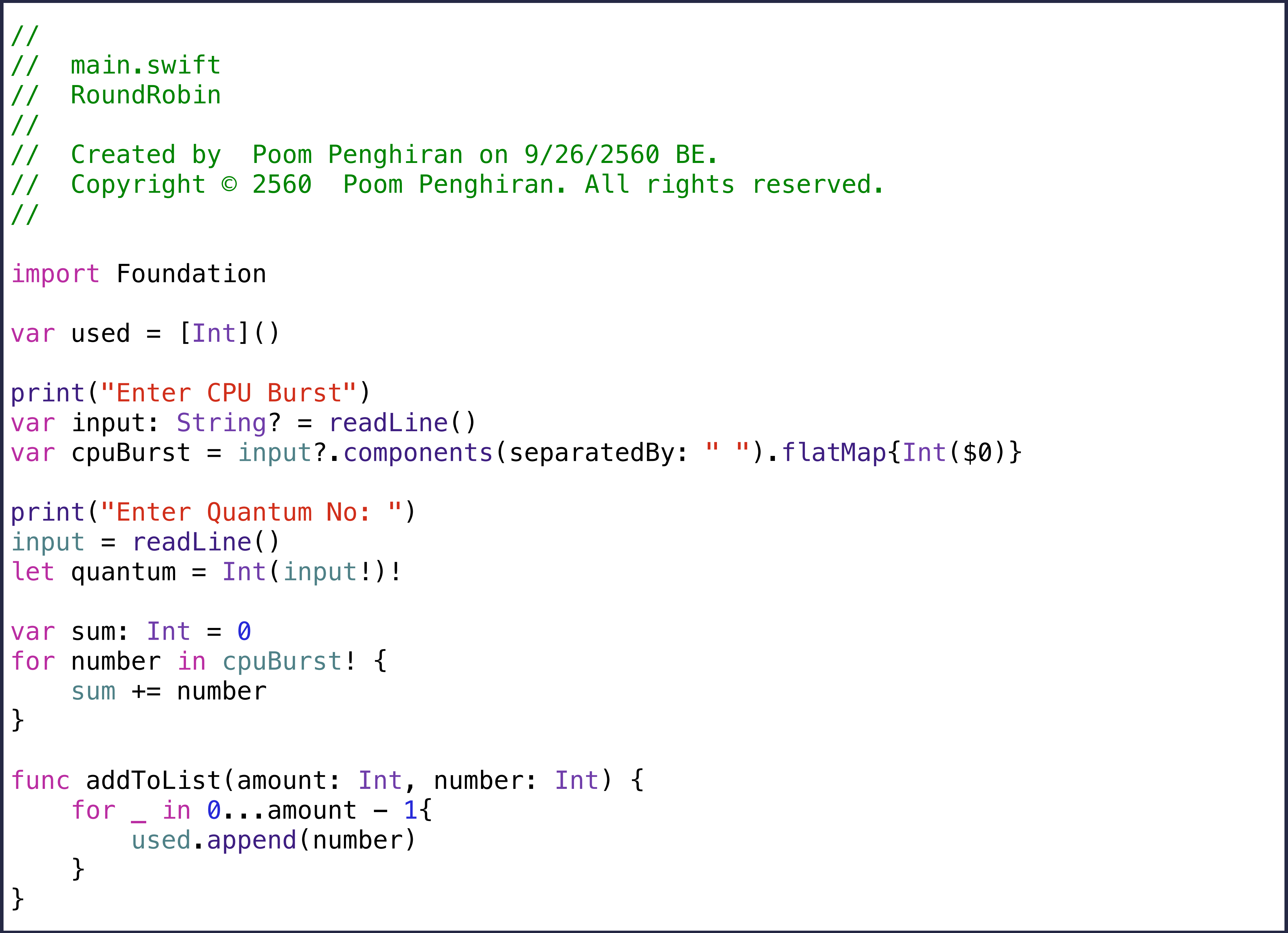
```swift
for i in cpuBurst {
    totalTime += i
}

func findLowest() -> Int{
    if jobList.count == 1 {
        return 0
    }
    let lowestKey = jobList.min()
    let location = jobList.index(of: lowestKey!)
    return location!
}

for i in 0...totalTime {
    if i < noProcess {
        jobList.append(priorityList[i])
    }
    let lowestIndex = findLowest()
    //fn find shortest
    if cpuBurst[lowestIndex] != 0 {
        used.append(lowestIndex)
        cpuBurst[lowestIndex] = cpuBurst[lowestIndex] - 1
        finishedTime[lowestIndex] = i + 1
    }

    //check if zero then record finish time
    if cpuBurst[lowestIndex] == 0 {
        jobList[lowestIndex] = 999
    }
}
```

Reference - Priority Scheduling



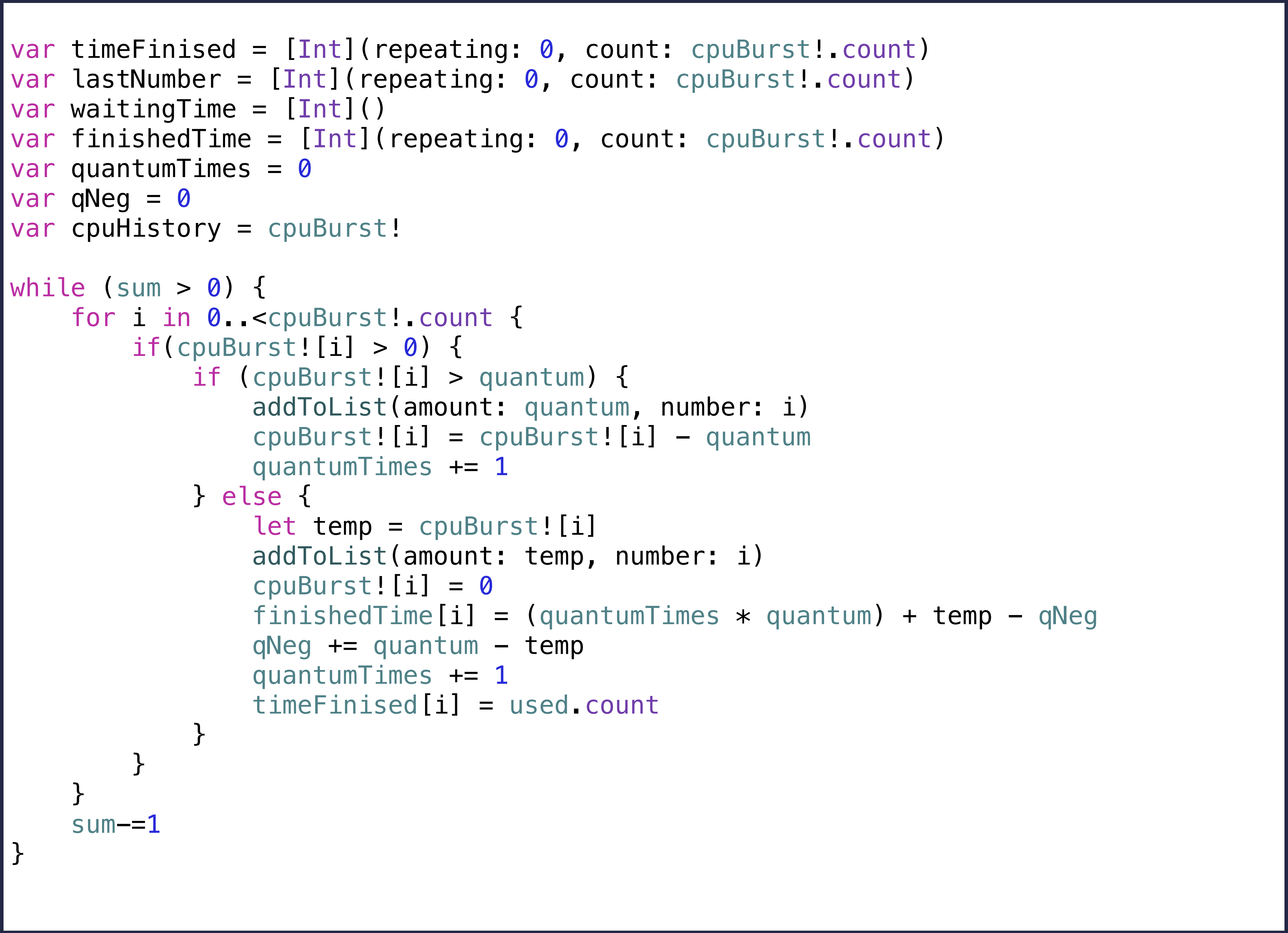
```swift
func calculateWaitingTime() -> Double {
    for i in 0..<noProcess {
        waitingTime[i] = (finishedTime[i] - cpuBackUp[i] - i)
    }
    var sum: Double = 0
    for i in waitingTime {
        sum += Double(i)
    }
    return sum / Double(noProcess)
}

print(used)
print("Average Waiting Time: \(calculateWaitingTime())")
```

Reference - Round-Robin



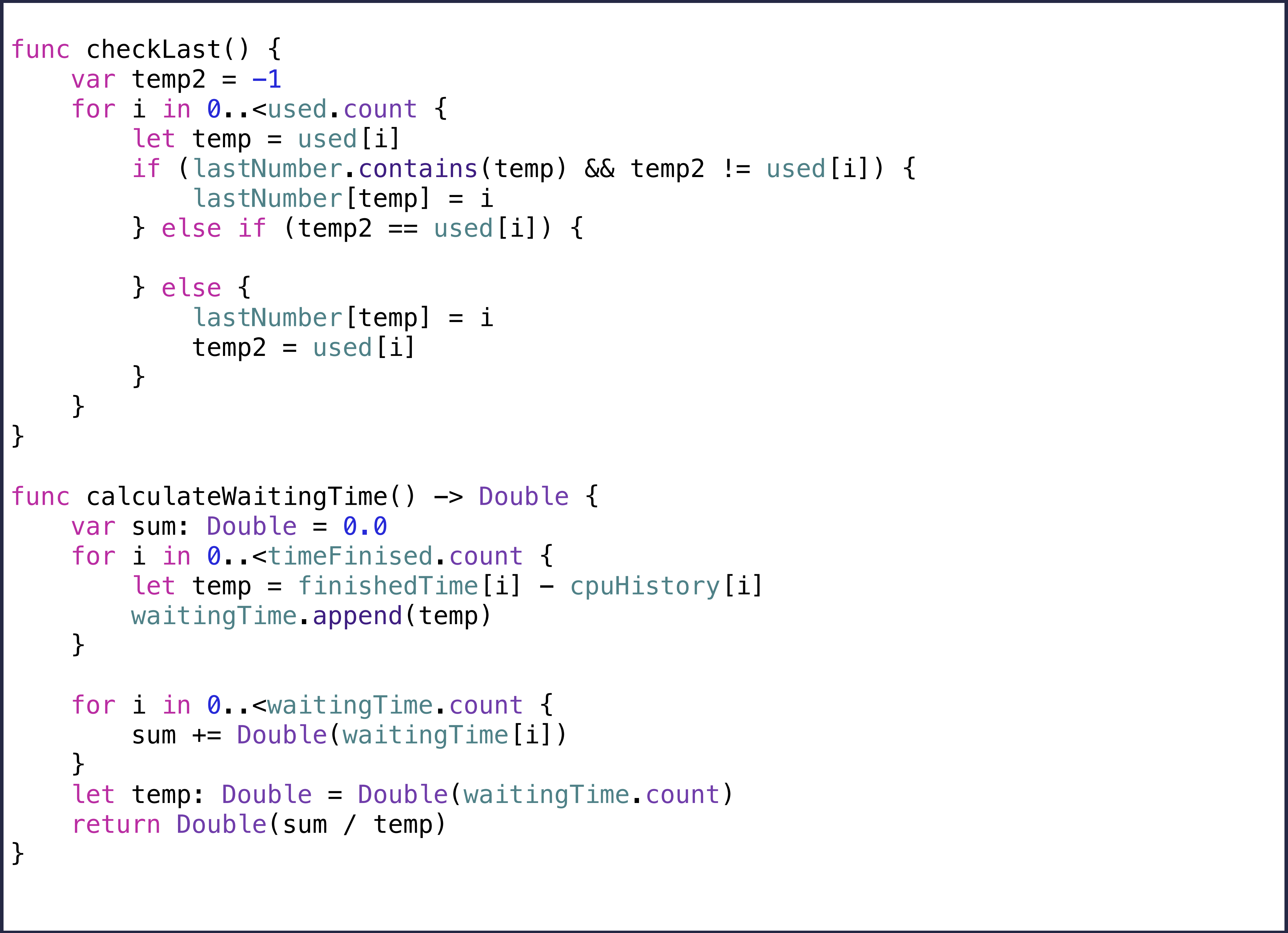
```swift
//
//  main.swift
//  RoundRobin
//
//  Created by  Poom Penghiran on 9/26/2560 BE.
//  Copyright © 2560  Poom Penghiran. All rights reserved.
//

import Foundation

var used = [Int]()

print("Enter CPU Burst")
var input: String? = readLine()
var cpuBurst = input?.components(separatedBy: " ").flatMap{Int($0)}

print("Enter Quantum No: ")
input = readLine()
let quantum = Int(input!)!

var sum: Int = 0
for number in cpuBurst! {
    sum += number
}

func addToList(amount: Int, number: Int) {
    for _ in 0...amount - 1{
        used.append(number)
    }
}
```

Reference - Round-Robin



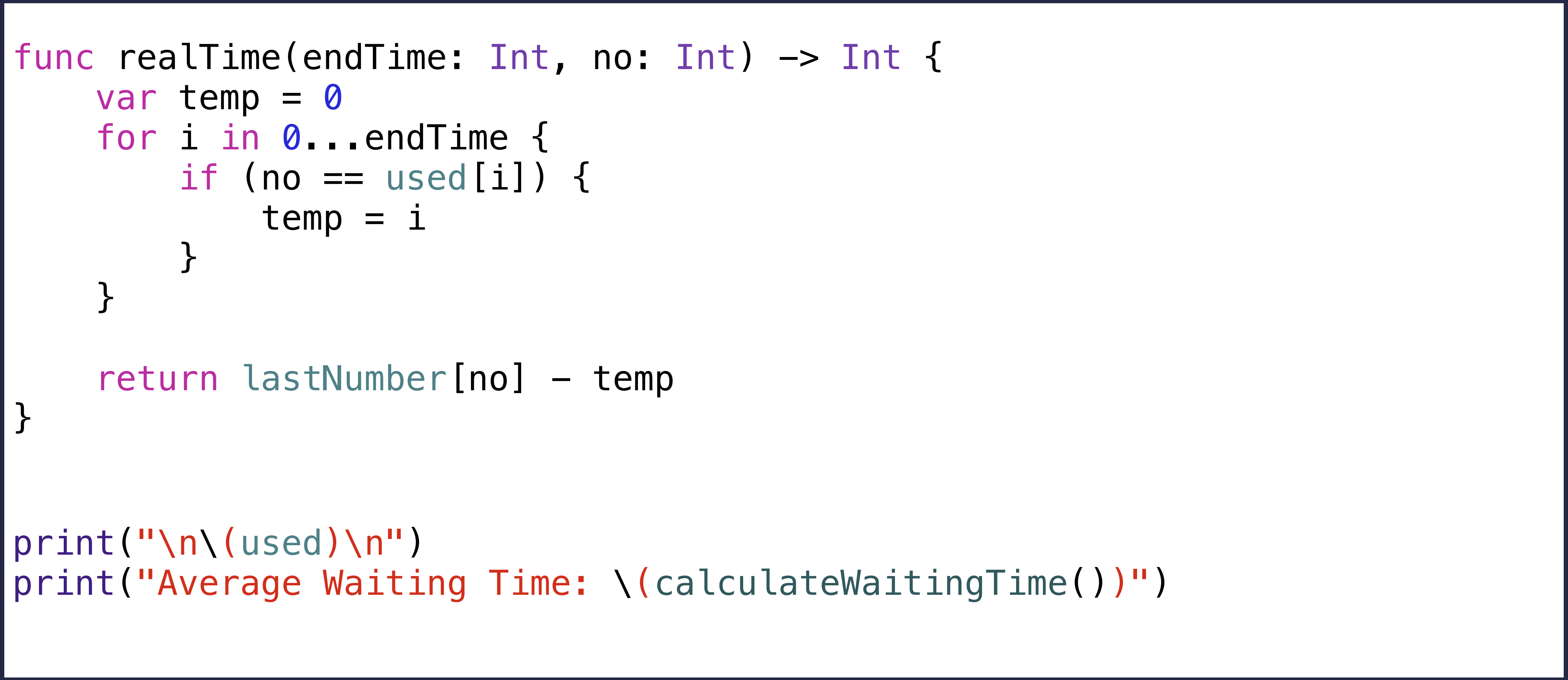
```
var timeFinised = [Int](repeating: 0, count: cpuBurst!.count)
var lastNumber = [Int](repeating: 0, count: cpuBurst!.count)
var waitingTime = [Int]()
var finishedTime = [Int](repeating: 0, count: cpuBurst!.count)
var quantumTimes = 0
var qNeg = 0
var cpuHistory = cpuBurst!

while (sum > 0) {
    for i in 0..<cpuBurst!.count {
        if(cpuBurst![i] > 0) {
            if (cpuBurst![i] > quantum) {
                addToList(amount: quantum, number: i)
                cpuBurst![i] = cpuBurst![i] - quantum
                quantumTimes += 1
            } else {
                let temp = cpuBurst![i]
                addToList(amount: temp, number: i)
                cpuBurst![i] = 0
                finishedTime[i] = (quantumTimes * quantum) + temp - qNeg
                qNeg += quantum - temp
                quantumTimes += 1
                timeFinised[i] = used.count
            }
        }
    }
    sum-=1
}
```

Reference - Round-Robin



```
func checkLast() {
    var temp2 = -1
    for i in 0..<used.count {
        let temp = used[i]
        if (lastNumber.contains(temp) && temp2 != used[i]) {
            lastNumber[temp] = i
        } else if (temp2 == used[i]) {

        } else {
            lastNumber[temp] = i
            temp2 = used[i]
        }
    }
}

func calculateWaitingTime() -> Double {
    var sum: Double = 0.0
    for i in 0..<timeFinised.count {
        let temp = finishedTime[i] - cpuHistory[i]
        waitingTime.append(temp)
    }

    for i in 0..<waitingTime.count {
        sum += Double(waitingTime[i])
    }
    let temp: Double = Double(waitingTime.count)
    return Double(sum / temp)
}
```

Reference - Round-Robin



```swift
func realTime(endTime: Int, no: Int) -> Int {
    var temp = 0
    for i in 0...endTime {
        if (no == used[i]) {
            temp = i
        }
    }

    return lastNumber[no] - temp
}


print("\n\(used)\n")
print("Average Waiting Time: \(calculateWaitingTime())")
```