一、适用人群

本教程适用于刚开始学编程和对nim语言来说是新手的。

二、nim安装

主要可参考官网地址，一般用choosenim来安装和控制nim版本成功率较高，且安装nim时需要链接外网。

三、值命名

1. 变量

Nim是静态类型语言，任何值使用之前都需要先声明。

其中可变变量的赋值方式如下：

```
var <name>:<type>
var <name>:<type> = <value>

var a:int          #仅声明不赋值
var v:int = 123    #声明同时赋值
var a = 123        #直接赋值，自动推断类型
```

另外可以进行批量赋值，格式如下(两个空格缩进，而非4个空格的Tab制表符)：

```
var
  a = 123
  b = 456
  c = 7879
```

可变变量的值是可以改变的，但改变前后类型必须一致！

2. 不可变赋值

(1) const

用const声明的常量值必须在编译之前就知道，如下示例：

```
const g = 35
g = 27              #错误，const声明的值不可变

var h = 5
const i = h +7      #错误，变量h在编译时不求值，常量值i在编译时无法获知，将导致错误
```

（2）let

用let声明的值在编译时可以不知道，但一旦被赋值后，就不可修改。

```
let j = 35
j = 12            #错误，let声明的值确定后就不可修改


var k = 5
let l = k +7      #正确，let可以在编译时暂时不知道它的值时
```

四、基本数据类型

1. 整数

整数时没有小数部分和小数点的数字：

```
32
-174
0
10_000_000    #这也是整数，_ 将被编译器忽略
```

整数运算规则（加减乘得到的是整数，除得到得是浮点数，div可以整除，mod取模）：

```
let
  a = 11
  b = 4

echo "a + b = ", a + b
echo "a - b = ", a - b
echo "a * b = ", a * b
echo "a / b = ", a / b
echo "a div b = ", a div b
echo "a mod b = ", a mod b
```

输出如下：

```
a + b = 15
a - b = 7
a * b = 44
a / b = 2.75
a div b = 2
a mod b = 3
```

2. 浮点数

浮点数仅支持4种基本运算法则：

```
let
  c = 6.75
  d = 2.25

echo "c + d = ", c + d
echo "c - d = ", c - d
echo "c * d = ", c * d
echo "c / d = ", c / d
```

输出如下：

```
c + d = 9.0
c - d = 4.5
c * d = 15.1875
c / d = 3.0
```

不同类型的值不可进行数学运算：

```
let
  e = 5
  f = 23.456

echo e + f   # error
```

所以先要进行类型转换：

```
let
  e = 5
  f = 23.987

echo float(e)
echo int(f)         #浮点数转整数，只是删除小数位，并没有执行四舍五入操作

echo float(e) + f
echo e + int(f)
```

输出如下：

```
5.0
23
28.987
28
```

3. 字符类型

char即为字符类型，单个字符用单引号包裹：

```
let
  h = 'z'
  i = '+'
  j = '2'
  k = '35' # 错误，只能单个字符
  l = 'xy' # 错误，只能单个字符
```

4. 字符串

string字符串类型应包裹在双引号内：

```
let
  m = "word"
  n = "A sentence with interpunction."
  o = ""
  p = "32"
  q = "!"
```

特殊字符：

```
\n  #换行
\t  #制表符
\\  #反斜杠
```

可见示例：

```
echo "some\\nim\\tips"
echo r"some\nim\tips"  #r"..."  将以原始字符串输出

输出：
some\nim\tips
some\nim\tips
```

字符串连接可以使用add或&符号：

```
var
  p = "abc"
  q = "xy"
  r = 'z'

p.add("def")
echo "p is now: ", p

q.add(r)
echo "q is now: ", q

echo "concat: ", p & q

echo "p is still: ", p
echo "q is still: ", q
```

输出为：

```
p is now: abcdef
q is now: xyz
concat: abcdefxyz
p is still: abcdef
q is still: xyz
```

5. 布尔值

bool，只能为true或false：

```
let
  g = 31
  h = 99

echo "g is greater than h: ", g > h
echo "g is smaller than h: ", g < h
echo "g is equal to h: ", g == h
echo "g is not equal to h: ", g != h
echo "g is greater or equal to h: ", g >= h
echo "g is smaller or equal to h: ", g <= h
```

输出为：

```
g is greater than h: false
g is smaller than h: true
g is equal to h: false
g is not equal to h: true
g is greater or equal to h: false
g is smaller or equal to h: true
```

字符串比较：

```
let
  i = 'a'
  j = 'd'
  k = 'Z'        #所有大写字母都在小写字母之前

echo i < j
echo i < k

let
  m = "axyb"
  n = "axyz"
  o = "ba"
  p = "ba "

echo m < n       #字符串都是逐字符比较，在前面的小于在后面的
echo n < o
echo o < p
```

输出如下：

```
true
false
true
true
true
```

还有逻辑运算：

```
echo "T and T: ", true and true  #与
echo "T and F: ", true and false
echo "F and F: ", false and false
echo "---"
echo "T or T: ", true or true    #或
echo "T or F: ", true or false
echo "F or F: ", false or false
echo "---"
echo "T xor T: ", true xor true  #异或，两个不同才是对的
echo "T xor F: ", true xor false
echo "F xor F: ", false xor false
echo "---"
echo "not T: ", not true         #非
echo "not F: ", not false
```

输出如下：

```
T and T: true
T and F: false
F and F: false
---
T or T: true
T or F: true
F or F: false
---
T xor T: false
T xor F: true
F xor F: false
---
not T: false
not F: true
```

五、控制流

1. if语句

很简单：

```
let
  a = 11
  b = 22
  c = 999

if a < b:  #true执行
  echo "a is smaller than b"
  if 10*a < b:  #false不执行
    echo "not only that, a is *much* smaller than b"
```

加上else：

```
let
  d = 63
  e = 2.718

if d < 10:
  echo "d is a small number"
else:
  echo "d is a large number"
```

加上elif：

```
let
  f = 3456
  g = 7

if f < 10:
  echo "f is smaller than 10"
elif f < 100:
  echo "f is between 10 and 100"
elif f < 1000:
  echo "f is between 100 and 1000"
else:
  echo "f is larger than 1000"
```

2.case

示例1：

```
case x
of 5:
  echo "Five!"
of 7:
  echo "Seven!"
of 10:
  echo "Ten!"
else:
  echo "unknown number"  #其他情况执行这一句
```

示例2：

```
case h
of 'x':
  echo "You've chosen x"
of 'y':
  echo "You've chosen y"
of 'z':
  echo "You've chosen z"
else: discard    #case语句必须包含所有可能的情况，如果我们不关心其他情况，则可以用
                   #这一句代替，没有它则代码无法编译
```

每个分支多种情况：

```
let i = 7

case i
  of 0:
    echo "i is zero"
  of 1, 3, 5, 7, 9:
    echo "i is odd"
  of 2, 4, 6, 8:
    echo "i is even"
  else:
    echo "i is too large"
```

六、循环

1. for循环

对可迭代类型进行迭代：

```
for n in 5 .. 9:   #5到9，包括9
  echo n

echo ""

for n in 5 ..< 9:  #5到9，不包括9
  echo n
```

带步长的迭代：

```
for n in countup(0, 16, 4):  #0到16，包括16，间隔4，
  echo n


for n in countdown(4, 0):    #倒数 4到0
  echo n

echo ""

for n in countdown(-3, -9, 2):  #倒数-3到-9，步长一定为正数
  echo n
```

字符串遍历

```
let word = "alphabet"

for letter in word:
  echo letter
```

计数器：

```
for i, letter in word:
  echo "letter ", i, " is: ", letter
```

输出为：

```
letter 0 is: a
letter 1 is: l
letter 2 is: p
letter 3 is: h
letter 4 is: a
letter 5 is: b
letter 6 is: e
letter 7 is: t
```

2. while循环

示例：

```
var a = 1

while a*a < 10:
  echo "a is: ", a
  inc a          # a递增1，类似

echo "final value of a: ", a
```

3. break continue

break退出循环：

```
var i = 1

while i < 1000:
  if i == 3:
    break
  echo i
  inc i
```

continue跳过当前本次循环：

```
for i in 1 .. 8:
  if (i == 3) or (i == 6):
    continue
  echo i
```

七、容器

1. 数组 array

数组内的元素必须类型相同，且数组的长度固定，必须在编译时就已知：

```
var
  a: array[3, int] = [5,6,7]
  b = [5, 7, 9]          #自动推断类型
  c = []  # error        #无法推断数组的长度和类型，则会报错
  d: array[7, string]  #空数组
```

必须在编译时知道数组长度，所以let类型数值不可用：

```
const m = 3
let n = 5

var a: array[m, char]
var b: array[n, char] # 错误，let在编译时不可
```

2. 序列 sequence

sequence的元素也要同类型，但长度可变，在编译时不需要知道长度：

```
var
  e1: seq[int] = @[]  #空seq
  f = @["abc","def"]  #自动推断类型
  e = newSeq[int]()   #过程调用生成空seq
```

seq添加及合并：

```
var
  g = @['x', 'y']
  h = @['1', '2', '3']

g.add('z')
echo g

h.add(g)
echo h
```

输出结果如下：

```
@['x', 'y', 'z']
@['1', '2', '3', 'x', 'y', 'z']
```

获取seq长度：

```
var i = @[9, 8, 7]
echo i.len

i.add(6)
echo i.len
```

3. 索引和切片

索引：

```
let j = ['a', 'b', 'c', 'd', 'e']

echo j[1]   #前数第二个 b
echo j[^1]  #倒数第一个 e
```

切片：

```
echo j[0 .. 3]  # @[a, b, c, d]
echo j[0 ..< 3] # @[a, b, c]
```

索引切片赋值：

```
var
  k: array[5, int]
  l = @['p', 'w', 'r']
  m = "Tom and Jerry"

for i in 0 .. 4:  #索引赋值
  k[i] = 7 * i
echo k

l[1] = 'q'        #索引赋值
echo l

m[8 .. 9] = "Ba"  #切片赋值，结果为89位置两个字符换为Ba=》Tom and Barry
echo m
```

4. 元组 tuple

元组可以收纳异构元素，但元组长度固定：

```
let n = ("Banana", 2, 'c')
echo n  # (Field0: "Banana", Field1: 2, Field2: 'c')
```

元组字段可以命名：

```
var o = (name: "Banana", weight: 2, rating: 'c')

o[1] = 7
o.name = "Apple"
echo o  #(name: "Apple", weight: 7, rating: 'c')
```

八、过程

声明过程的格式：

```
proc <name>(<p1>: <type1>, <p2>: <type2>, ...): <returnType>
```

例如：

```
proc findMax(x: int, y: int): int =
  if x > y:
    return x
  else:
    return y
  # this is inside of the procedure
# this is outside of the procedure
```

以上即声明了一个findMax过程，传入x，y两个参数，返回int值。

一个比较有意思的链式调用语法：

```
proc plus(x, y: int): int =
  return x + y

proc multi(x, y: int): int =
  return x * y

let
  a = 2
  b = 3
  c = 4

echo a.plus(b) == plus(a, b)  #两种调研方法都行
echo c.multi(a) == multi(c, a)


echo a.plus(b).multi(c)  #链式调用
echo c.multi(b).plus(a)
```

想返回的值也可以直接赋值给result变量，这是一个默认存在的变量：

```
proc findBiggest(a: seq[int]): int =
  for number in a:
    if number > result:
      result = number  #result就是返回的值，即使没有return
  # the end of proc

let d = @[3, -5, 11, 33, 7, -15]
echo findBiggest(d)
```

各类型默认值：

```
int :  0
string : ""
seq : @[]
```

前向声明，先声明再定义过程：

```
proc plus(x, y: int): int

echo 5.plus(10)

proc plus(x, y: int): int =
  return x + y
```

九、模块

1. 导入模块

```
import strutils         #导入

let
  a = "My string with whitespace."
  b = '!'

echo a.split()         #不需要命名空间前缀就直接使用了
echo a.toUpperAscii()
echo b.repeat(5)
```

2. 创建模块

first.nim:

```
proc plus*(a, b: int): int =   #带*可被外部访问
  return a + b

proc minus(a, b: int): int =
  return a - b
```

second.nim:

```
import first

echo plus(5, 10)
echo minus(10, 5) # error, 不带*不可外部访问
```

子目录导入：

```
.
├── myOtherSubdir
│   ├── fifthFile.nim    #import myOtherSubdir / [fourthFile, fifthFile]
│   └── fourthFile.nim
├── mySubdir
│   └── thirdFile.nim
├── firstFile.nim
└── secondFile.nim
```

十、用户交互

```
echo "Please enter your name:"
let name = readLine(stdin)  #字符串输入

echo "Hello ", name, ", nice to meet you!"
```

```
import strutils

echo "Please enter your year of birth:"
let yearOfBirth = readLine(stdin).parseInt()  #数字输入

let age = 2018 - yearOfBirth

echo "You are ", age, " years old."
```