0628 Python / AI Programming Practice

Tutor: Nanyang Ye
Note Taker: Y. Qiu

In [ord]

In [*]: Running!

Inputs and Outputs

In [43]:

```
#    Example 01
#    Input and Eval
s = input("Please input your name.")
print(s)
print(type(s))
print(s[1])

a = eval(' 3 * 7 ')
print(a)
a = eval(" 3 * 7 ")
print(a)
```

```
Please input your name.qyh
qyh
<class 'str'>
y
21
21
```

In Python, ' and " is equivalent.

In [24]:

```
#    Example 02 (the default property of input is string)
a = input()
print(a)
print(type(a))

b = int(input())
print(b)
print(type(b))
```

```
57
57
<class 'str'>
23
23
<class 'int'>
```

In [1]:

```
print("Hey there.\nYou got 1078 bugs.")

print("""
Hey there.
You got a lot of bugs.
""")

print("[END]")

print("a",end="")
print("b")
```

```
Hey there.
You got 1078 bugs.

Hey there.
You got a lot of bugs.

[END]
ab
```

Assignments

In [45]:

```
a = 1
b = 2.0
name = "world"
sent = 'world'

print(type(a),type(b),type(name),type(sent))
```

```
<class 'int'> <class 'float'> <class 'str'> <class 'str'>
```

In [46]:

```
a, b, c = 1, 2.0, "world"
print(a,' ',b,' ',c)
```

```
1   2.0   world
```

Logic Expressions

In [48]:

```
print(1 != 2)
print(1 != 1)
```

```
True
False
```

In [49]:

```
print(3>5)
```

```
False
```

In [50]:

```
2 == 2
```

Out[50]:

```
True
```

In [8]:

```
3 <= 5
```

Out[8]:

```
True
```

In [9]:

```
3 % 2==0
```

Out[9]:

```
False
```

In [11]:

```
1+ False + 1.2
```

Out[11]:

```
2.2
```

In [12]:

```
True / 1.2 + 1
```

Out[12]:

1.8333333333333335

In [13]:

```
-5 % 3
```

Out[13]:

1

In [2]:

```
(2 > 1) and (3 != 4)
```

Out[2]:

True

In [3]:

```
(2 > 1) or (3 != 3)
```

Out[3]:

True

In [4]:

```
not (3 != 3)
```

Out[4]:

True

Basic Computations

In [59]:

```
a = 9
b = 2
a *= b
print("*",a)

a += b
print("+",a)

a -= b
print("-",a)

print("/",a/7)          # division; default: float
print("//",a//7)        # division (result as integer)

print(eval('-5 // 7'))
print(eval('-5 % 7'))

a %= 7
print("%",a)

a **= 3
print("**",a)           # power
```

```
* 18
+ 20
- 18
/ 2.5714285714285716
// 2
-1
2
% 4
** 64
```

Binary Computations

In [1]:

```
a = 14       #0000 1110
b = 3        #0000 0011

print(a & b)    #0000 0010    AND

print(a | b)    #0000 1111    OR

print(~a)       #0001    NOT

print(a ^ b)    #1101    XOR

a = a << 1
print("<<",a)

a = a >> 2
print(">>",a)
```

```
2
15
-15
13
<< 28
>> 7
```

Case Structure

if - else

Notice that there is a ":" after the condition expression.

In [22]:

```
#     if
a = 5
if (a % 2 == 0):
    print("even")
else:
    print("odd")
```

```
odd
```

if - elif - else

In [23]:

```
#     if - elif
a = 5
if (a % 2 ==0):
    print("even")
elif (a % 3 ==0):
    print("something.")
else:
    print("anything.")
```

```
anything.
```

Loop Structure

while (- else) loop

```
    while (condition): #do something

    while (condition):
        # do something
    else:
        # do something
```

In [5]:

```
sum = 1
a = int(input())
n = int(input())
while (n > 0):
    if (n & 1):
        sum *= a
    sum *= sum
    n = n >> 1

print(sum)
```

```
20
20
10240000000000
```

for - loop

```
    for i in range(n):     #0..(n-1)
        #do something

    for i in ['a','b','c']:
        #do something
```

In [30]:

```
sum = 0
for i in range(101):
    sum += i
print(sum)
```

5050

In [37]:

```
print("2是素数")
n = int(input())
i = 3
while (i<n):
    j = 2
    while (j<=n/i):
        if (i%j==0): break
        j = j+1
    if j > n/i: print(i,"是素数")
    i += 2
```

2是素数
30
7 是素数
11 是素数
13 是素数
15 是素数
17 是素数
19 是素数
21 是素数
23 是素数
25 是素数
27 是素数
29 是素数

In [ ]:

```
while True :
    year = eval(input("请输入年份"))
    if ((year % 100 != 0 and year % 4 == 0) or (year % 400 == 0)):
        print(year,"是闰年")
    else:
        print(year,"不是闰年")
```

请输入年份2021
2021 不是闰年
请输入年份2020
2020 是闰年
请输入年份2030
2030 不是闰年
请输入年份1990
1990 不是闰年
请输入年份1900
1900 不是闰年
请输入年份2048
2048 是闰年

Functions

Definition of Function(s)

```
def <Name>(parameters):

    #do something

    return  # return None, can be omitted
    # return something
```

No need to claim the class of parameters nor the class of the return vale

In [32]:

```
def max(a,b):
    if a>b: return a
    else: return b


a = 1
b = 2
print(max(a,b))

a = "a"
b = "b"
print(max(a,b))

a = 3.8
b = 4
print(max(a,b))

a = ord("a")    # comparison with a = "a"  #(error)
b = 4
print(max(a,b))
```

```
2
b
4
97
```

What will happen if there are multiple "return"s in the same function?

-- Jump out at the first "return".

In [13]:

```
def Add(a, b):
    c = a+b
    return c
    return a

a = 2
b = 500
print(Add(a, b))
```

```
502
```

In [14]:

```
def AbaAba(a, b):
    print(a)

b = AbaAba(3, 500)
print(b)
print(type(b))
```

```
3
None
<class 'NoneType'>
```

"None" is a specifically defined (object-oriented) class.

Parameters

position parameters

keyword parameters

parameters with default argument

In [19]:

```
def func(a, b):
    print("a = ", a)
    print("b = ", b)
    return

func("1", "2")            # Position Parameters

func("1", b="2")          # Keyword Parameteres (b)       # Recommended!
func(b="1", a="2")        # Keyword Parameteres (a and b)
```

```
a =  1
b =  2
a =  1
b =  2
a =  2
b =  1
```

In [27]:

```
def func(a, b, c=5):                # like C++, the non-default argument(s) must be leading all defaul
    print("a = ", a)
    print("b = ", b)
    print("c = ", c)
    return

func("1", 3, "a")                   # parameters with defalut argument can be assigned a value of a tot
print('\n')

func(1, "3")
print('\n')

func(b=1, c="@", a='3')
func(b=1, a='3')
print('\n')

func(1, b="#")                     # Recommended Style
func(1, b="#", c="*")              # Recommended Style
```

```
a =  1
b =  3
c =  a


a =  1
b =  3
c =  5


a =  3
b =  1
c =  @
a =  3
b =  1
c =  5


a =  1
b =  #
c =  5
a =  1
b =  #
c =  *
```

Local and Global Variables

The action scope of local and global variables. (Omitted)

Parameters with Determined Type

```
def <func>(parameters:type_name)->type_name:
```

In [4]:

```
def max(a:int, b:int, c:int)->str:
    a = a+b+c
    return (str(a)+str(b)+str(c))

print(max(3,5,7))
print(max(3.2,5,7))
```

```
1557
15.257
```

Non-determined Parameters（不定长参数）

```
def <func>(parameters, *val):

    #do something
```

"*val" is a tuple containing all non-determined parameters.

Another way of implementing non-determined parameters is to use "**", i.e.

==========

```
def <func>(parameteres, **val):

    # do something
```

where val is a dictionary containing all non-determined parameters.

In [34]:

```
def Func(a,*val):
    print("a = ",a)
    for var in val:
        print(var)
    return

Func("@",1,"a",'steste',7.08213423,['a',1,3.0])
```

```
a =  @
1
a
steste
7.08213423
['a', 1, 3.0]
```

The running time of transferring different type into functions.

In [37]:

```
def Func(a,*val):
    print("a = ",a)
    return

import time

start_t = time.time()
Func(5,6,7)
end_t = time.time()
print (end_t-start_t)

start_t = time.time()
Func([5,6,7])
end_t = time.time()
print (end_t-start_t)
```

```
a =  5
0.0
a =  [5, 6, 7]
0.0
```

Return Multiple Values

Return as a tuple.

A tuple : (value, value, value, value)

The values in a tuple can be of the same class.

In [40]:

```
def Add(a,b):
    return a+b, a-b, a*b, a/b, "@"

a = 5
b = 3
print(Add(a,b))
```

```
(8, 2, 15, 1.6666666666666667, '@')
```

Anonymous Functions (Lambda Function)

```
lambda <parameters>:<expression>
```

In [109]:

```
c = lambda a,b,c=5 : a+b+c
print(c(1,2))
print(c(1,2,8))

print((lambda a,b,c=1 : a*b*c )(1,2,3))
```

```
8
11
6
```

Module

```
import [something]
import [something] as sth

from [Something] import [something in Something]
from [Something] import [something in Something] as Sth
```

We can import modules from the other file under the same folder.

```
e.g.

-- demo
    |-- alpha.py
    |__ beta.py
```

[IN ALPHA.PY]

```
import beta
from beta import afunctioninbeta as func
```

List

A list of different types of data.

String("str") is a special kind of list.

Initialization

```
a = list()        # an empty list
b = list([2,3,4])   # a list of [2,3,4]

c = [2,3,4]       # In fact, create an anonymous list "[2,3,4]" and assign it to "c".    #
 Low Efficiency
```

【index: 0..n-1】

In [47]:

```
lis = [2,3.4,"@","strstr"]

print(len(lis))
print(lis)
print(lis[0],' ',lis[1],' ',lis[2],' ',lis[3])
print(lis[-1])                                  # The last element in the list.
#print(lis[-10])          # Error: Index out of range
                          # ↑ Infiltration Test.    e.g. Password
```

```
4
[2, 3.4, '@', 'strstr']
2   3.4   @   strstr
strstr
```

Selection of Elements in a List

```
[a:b] select all elements from listname[a] to listname[b]

    Comprehension: (a elements) [:<the elements we want>] (the (b+1)-st element)
    n fact: listname[a] ~ listname[b]    ** Notice that index:[0..len-1]

[:b] the first b elements
[a:] the last (len-a) elements
```

In [50]:

```
lis = [1,2,3,4,5,6,7]

print(lis[1:3])
print(lis[:2])
print(lis[2:])
```

```
[2, 3]
[1, 2]
[3, 4, 5, 6, 7]
```

In [51]:

```
list1 = [1,2]
list2 = [3,4]

list3 = list1+list2

print(list3)
```

```
[1, 2, 3, 4]
```

In [112]:

```
list1 = [x for x in range(1,21)]
print(list1)

list1 = [x for x in range(20)]
list2 = list(range(0,20))

print(list1)
print(list2)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Multi-Dimension List

In [103]:

```
list1 = [[1,2],[3,4]]
print(list1)
```

```
[[1, 2], [3, 4]]
```

Tuples, Sets and Dictionaries

Tuples

"Constants". Fixed elements. (element1, element2, element3)

More efficient than List.

In [95]:

```
tuple1 = ()   #empty tuple
tuple0 = tuple()
tuple2 = (1,2,3)
tuple3 = tuple([x for x in range(1,21)])     # turning a list into tuple
tuple4 = tuple(range(1,21))

print(tuple1)
print(tuple0)
print(tuple2)
print(tuple3)
print(tuple4)
```

```
()
()
(1, 2, 3)
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20)
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20)
```

Sets

Sets. Non-redundant (no repeating elements) and non-ordered. {element1, element2, element3}

In [77]:

```
set1 = set()
set2 = {}
set3 = {(1,2,3)}
set4 = set([1,2,3])        # Turning a list into a set.
                           # Cannot use {[1,2,3]} to turn a list into a set, 'cause the order informa

print(set1)
print(set2)
print(set3)
print(set4)

# print(set4[2])         #ERROR

for i in set4:
    print(i)
```

```
set()
{}
{(1, 2, 3)}
{1, 2, 3}
1
2
3
```

In [82]:

```
set1 = {1,2,3}

set1.add(3)
print(set1)

set1.update({4,6})
print(set1)

set1.pop()          # randomly pick an element in the set and remove it
print(set1)
set1.discard(3)     # faster
print(set1)
set1.remove(4)
print(set1)
```

```
{1, 2, 3}
{1, 2, 3, 4, 6}
{2, 3, 4, 6}
{2, 4, 6}
{2, 6}
```

In [89]:

```
set1 = {1,2,3}
set2 = {3,4,5}

print(set1.issubset(set2))
print(set1.issuperset(set2))
print(set1 == set1)
print(set1 == set2)
print(set1 != set1)

print(set1.union(set2))              # Union
print(set1.intersection(set2))       # Intersection
print(set1.difference(set2))         # Difference
print(set1-set2)                     # Difference
```

```
False
False
True
False
False
{1, 2, 3, 4, 5}
{3}
{1, 2}
{1, 2}
```

Dictionaries

Key ---- Data.

{"key1":"element1","key":"element2"}

In [94]:

```
students = {"001":"John","002":"Hash","003":"S**t"}
EmptyDic = {}
# Dictionaries are created on the basis of set

print(students["001"])
del students["003"]        # remove a specific element
```

John