Biological data analysis (4)
Control flow

Aleksandras Voicikas
avoicikas@gmail.com

Flow control

- While, if, else, for, pass, continue
- and, not, or, is
- AND, XOR, SHIFT, OR, NOT

Overview

- Variables, indexing
- Operators
- Functions
- Built-in functions, methods
- Printing, r and f strings, special characters
- Plotting

```
1  def celsius_to_farenheit(celsius):
2      fahrenheit = (celsius * 9/5) + 32
3      kelvin = celsius + 273.15
4      return round(fahrenheit), round(kelvin)
5
6
7  output = celsius_to_farenheit(42)
8  output
9
10 (108, 315)
```

Basic operations

+	sum
−	difference
*	multiplication
**	to the power of
<	less
>	more
==	equal
!=	not equal
<=	less or equal
/	division producing floats
//	division producing integers
%	reminder of division

Logical operations

```
1   a = True
2   b = False
3
4   # and checks if both are true
5   print(a and b)
6
7   False
8
9   # or checks if one of choices is true
10  print(a or b)
11
12  True
13
14  # not reverses check
15  print(not a)
16
17  False
```

is

The difference between is and ==
is operator checks if both the operands refer to the same object (i.e., it checks if the identity of the operands matches or not). == operator compares the values of both the operands and checks if they are the same.

```
a = 256
b = 256
a is b
True


a = 257
b = 257
a is b
False

# 256 is an existing object but 257 isn't
```

When you start up python the numbers from -5 to 256 will be allocated. These numbers are used a lot, so it makes sense just to have them ready.

If statement

- if, elif, else
- Colon after every statement
- Identation !

```
1   x = int(input("Please enter an integer: "))
2   Please enter an integer: 5
3   if x < 0:
4       x = 0
5       print('Negative changed to zero')
6   elif x == 0:
7       print('Zero')
8   elif x == 1:
9       print('Single')
10  else:
11      print('More')
```

If statement

```
1  def convert_to_SI(value, type='temp'):
2      ''' supported types:
3              temp (fahrenheit input, celsius and kelvin outputs)
4              dist (inches input, meters output)'''
5      if type == 'temp':
6          celsius = (value - 32) * 5/9
7          kelvin = celsius + 273.15
8          return round(celsius), round(kelvin)
9      elif type == 'dist':
10         meters = value * 0.0245
11         return(meters)
12     else:
13         print('Type is not defined')
```

For statement

```
1  for x in [1,2,3]:
2      print(x)
3
4  1
5  2
6  3
```

For statement

```
1   values_in_fahrenheit = [0,44,56,788]
2   values_in_celsius= []
3   values_in_kelvin= []
4
5   for value in values_in_fahrenheit:
6       cel, kel = convert_to_SI(value)
7       values_in_celsius.append(cel)
8       values_in_kelvin.append(kel)
9
10
11  values_in_kelvin
12  [255, 280, 286, 693]
13
14  values_in_celsius
15  [-18, 7, 13, 420]
```

range()

range([start], stop[, step])

```
1  for i in range(4):
2      print(i)
3
4  0
5  1
6  2
7  3
8
9  In [1]: range(5, 10)
10 Out[1]: range(5, 10)
11    5, 6, 7, 8, 9
12
13 range(0, 10, 3)
14    0, 3, 6, 9
15
16 range(-10, -100, -30)
17   -10, -40, -70
18
19 list(range(4))
20 [0, 1, 2, 3]
```

for

```
1  for i in range(4):
2      print(i)
3      i = 10
4
5  0
6  1
7  2
8  3
```

Progress bar

```
1   from time import sleep
2   def progress(percent=0, width=30):
3       left = width * percent // 100
4       right = width - left
5       print('\r[', '#' * left, ' ' * right, ']',
6             f' {percent:.0f}%',
7             sep='', end='', flush=True)
8
9
10  for i in range(101):
11      progress(i)
12      sleep(0.1)
13
14  [############################    ] 87%
```

for with dictionaries

```
1  data = {'-18': '255', '7': '280'}
2  for k, v in data.items():
3      print(f'Celsius: {k}, Kelvin {v}')
4
5  Celsius: -18, Kelvin 255
6  Celsius: 7, Kelvin 280
```

enumerate

```
1  a = ['a', 'b', 'c'];
2  for i, v in enumerate(a):
3      print(i, v)
4
5  0 a
6  1 b
7  2 c
8
9  for i in range(len(a)):
10     print(i,a[i])
```

zip

To loop over two or more sequences at the same time, the entries can be paired with the zip() function.

```
values_in_celsius # -18 7 13 420
values_in_fahrenheit # 0 44 56 788

for q, a in zip(values_in_fahrenheit, values_in_celsius):
    print(f'Temperature in celsius {a} equals {q} in fahrenheit')

Temperature in celsius -18 equals 0 in fahrenheit
Temperature in celsius 7 equals 44 in fahrenheit
Temperature in celsius 13 equals 56 in fahrenheit
Temperature in celsius 420 equals 788 in fahrenheit
```

Reverse iteration

```
1  for i in reversed(range(1, 10, 2)):
2      print(i)
3
4  9
5  7
6  5
7  3
8  1
```

while statement

While is executing till we reach some value

```
1   a, b = 0, 1
2   while a < 10:
3       print(a)
4       a, b = b, a+b
5
6   0
7   1
8   1
9   2
10  3
11  5
12  8
```

Infinite loops

```
1  while True:
2      print('a')
3
4  import random
5  x = 0
6  while x<0.9:
7      x = random.random()
8      print(x)
9
10 0.21195697906981703
11 0.921747320805385
```

break and continue

The break statement breaks out of the innermost enclosing for or while loop. The continue statement continues with the next iteration of the loop.

```
34%10
4

for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, 'equals', x, '*', n//x)
            break
    else: # for loop condition else is evaluated if no break was encountered
        print(n, 'is a prime number')

2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

pass

pass does nothing

```
1 for i in range(4):
2     pass
```

Bitwise operations

&	AND	$x\&y$
|	OR	$x\|y$
~	NOT	$\sim x$
^	XOR	$x\hat{}\,y$
>>	right shift	$x >>$
<<	left shift	$x <<$

Binary system

Converting to decimal numbers:

1	1	1	1	1	1	1	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128 +	64 +	32 +	16 +	8 +	4 +	2 +	1 = 255

Binary system

0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Octal and Hexadecimal systems

Dec	Bin	Hex	Oct
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7

Dec	Bin	Hex	Oct
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

Usage

- Yes/no questions/options (flags, permissions)
- Communication (checksums, flow control...)
- Encryption, compression
- Graphics (redraw)

AND

x & y

Does a ”bitwise and”. Each bit of the output is 1 if the corresponding bit of x AND of y is 1, otherwise it’s 0.

0001
0010
0000

```
1 & 2
0
3 & 9
1
```

0011
1001
0001

OR

x | y

Does a ”bitwise or”. Each bit of the output is 0 if the corresponding bit of x AND of y is 0, otherwise it’s 1.

0001

0011

0011

```
1|3
3
```

0010

1001

1011

```
2|9
11
```

NOT

~ x

Returns the complement of x - the number you get by switching each 1 for a 0 and each 0 for a 1. This is the same as -x - 1.

```
1  ~4
2  -5
```

XOR

x ^ y

Does a ”bitwise exclusive or”. Each bit of the output is the same as the corresponding bit in x if that bit in y is 0, and it’s the complement of the bit in x if that bit in y is 1.

Used in checksums

0011

0010

0001

```
3 ^ 2
1
```

0011

1110

1101

```
3 ^ 14
13
```

Left shift

x » y

Returns x with the bits shifted to the right by y places. This is the same as //'ing x by 2**y.

1000

0100

```
1  8 >> 1
2  4
```

Right shift

x « y
Returns x with the bits shifted to the left by y places (and new bits on the right-hand-side are zeros). This is the same as multiplying x by 2**y.
0100
1000

```
4 << 1
8
```

Evaluation Nr. 1

Next week 2020-03-05

- Indexing, slicing variables
- Creating variables, using built-in methods (append, extend, pop, etc)
- Creating functions, evaluating formulas
- Flow control exercises
- Formatting and printing out information