Scientific Programming Practical 2

Introduction

Luca Bianco - Academic Year 2018-19
luca.bianco@fmach.it

Modules and Objects

Modules are text files with .py extension

```
python3 exercise1.py
```

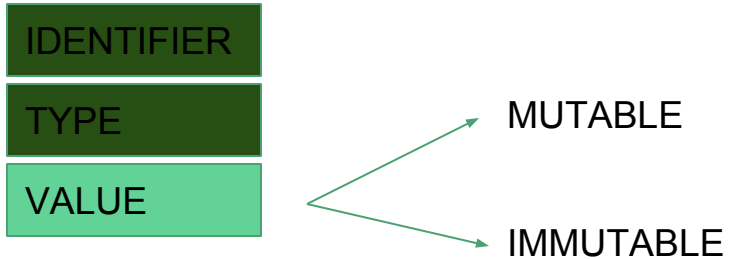
Import modules to use them

```
import math

A = math.sqrt(4)
print(A)
```

```
2.0
```

Objects

“Objects are Python’s abstraction for data. All data in a Python program is represented by objects or by relations between objects.”



Built-in data types

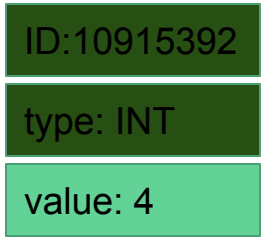
Type	Meaning	Domain	Mutable?
`bool`	Condition	`True`, `False`	No
`int`	Integer	$\{-2^{63}, \ldots, 2^{63}-1\}$ $\mathbb{Z}$	No
~~`long`~~	~~Integer~~	~~$\mathbb{Z}$~~	~~No~~
`long`	Integer	$\mathbb{Z}$	No
`float`	Rational	$\mathbb{Q}$ (more or less)	No
`str`	Text	Text	No
`list`	Sequence	Collections of things	Yes
`tuple`	Sequence	Collections of things	No
`dict`	Map	Maps between things	Yes

Variable assignment

What happens when we…

>>> sides = 4

A new object is created and the name ‘sides’ points to it




```
sides = 4
print( type(sides) )
print( id(sides) )
```

```
<class 'int'>
10915392
```

INT is **immutable**, therefore:

```
sides = 4 #a square
print ("value:", sides, " type:", type(sides), " id:", id(sides))
sides = 5 #a pentagon
print ("value:", sides, " type:", type(sides), " id:", id(sides))
```

```
value: 4  type: <class 'int'>  id: 10915392
value: 5  type: <class 'int'>  id: 10915424
```

Name of variables

You can choose the name you like but:

1. Can only contain A-Z, a-z, 0-9 or _
2. Cannot start with a number
3. Cannot be one of the reserved words

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

Integers

As one would expect...

```
a = 7
b = 4

a + b # 11
a - b # 3
a // b # integer division: 1
a * b # 28
a ** b # power: 2401
a / b # division 0.8333333333333334
type(a / b)
```

```
float
```

REMEMBER: Immutable

Their range is limited ONLY by the AVAILABLE memory

Booleans

Assume only values True and False

```
a = bool(1)
b = bool(0)
c = bool(72)
d = bool(-5)
t = int(True)
f = int(False)

print("a: ", a, " b: ", b, " c: ", c, " d: ", d , " t: ", t, " f: ", f)
```

```
a:  True  b:  False  c:  True  d:  True  t:  1  f:  0
```

Boolean algebra rules...

```
T = True
F = False

print ("T: ", T, " F:", F)

print ("T and F: ", T and F) #False
print ("T and T: ", T and T) #True
print ("F and F: ", F and F) #False
print ("not T: ", not T) # False
print ("not F: ", not F) # True
print ("T or F: ", T or F) # True
print ("T or T: ", T or T) # True
print ("F or F: ", F or F) # False
```

All numbers evaluate to **True, except 0**.

Reals

In python they are floating points (floats)

Example: Let's calculate the area of the center circle of a football pitch (radius = 9.15m) recalling that $area = Pi * R^2$:

```
In [11]: R = 9.15
         Pi = 3.1415926536
         Area = Pi*(R**2)
         print (Area)
```

```
263.02199094102605
```

Use parenthesis or remember precedence of operators...

`**`	Power (Highest precedence)
`+,-`	Unary plus and minus
`* / // %`	Multiply, divide, floor division, modulo
`+ -`	Addition and subtraction
`<= < > >=`	Comparison operators
`== !=`	Equality operators
`not or and`	Logical operators (Lowest precedence)

Strings

In python they are immutable objects to deal with text

```
S = "my first string, in double quotes"

S1 = 'my second string, in single quotes'

S2 = '''my third string is
in triple quotes
therefore it can span several lines'''

S3 = """my fourth string, in triple double-quotes
can also span
several lines"""

print(S, '\n') #let's add a new line at the end of the string with \n
print(S1,'\n')
print(S2, '\n')
print(S3, '\n')
```

```
my first string, in double quotes

my second string, in single quotes

my third string is
in triple quotes
therefore it can span several lines

my fourth string, in triple double-quotes
can also span
several lines
```

Strings

Escape special characters

\\	Backslash
\n	ASCII linefeed (also known as newline)
\t	ASCII tab character
\'	Single quote
\"	Double quote
\xxxx	Unicode character xxxx (hexadecimal)

```
myString = "This is how I \'quote\' and \"double quote\" things in strings"
print(myString)
```

```
This is how I 'quote' and "double quote" things in strings
```

```
print("Greek omega is: \u03C9")
```

```
Greek omega is: ω
```

Strings

Functions

Result	Operator	Meaning
`int`	`len(str)`	Return the length of the string
`str`	`str + str`	Concatenate two strings
`str`	`str * int`	Replicate the string
`bool`	`str in str`	Check if a string is present in another string
`str`	`str[int]`	Read the character at specified index
`str`	`str[int:int]`	Extract a sub-string

Strings

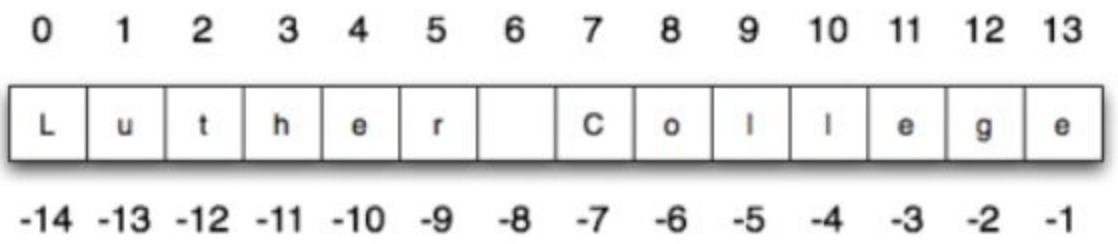
Result	Operator	Meaning
int	len(str)	Return the length of the string
str	str + str	Concatenate two strings
str	str * int	Replicate the string
bool	str in str	Check if a string is present in another string
str	str[int]	Read the character at specified index
str	str[int:int]	Extract a sub-string

Example A tandem repeat is a short sequence of DNA that is repeated several times in a row. Let's create a string representing the tandem repeat of the motif "ATTCG" repeated 5 times. What is the length of the whole repetitive region? Is the motif "TCGAT" (m1) present in the region? The motif "TCCT" (m2)? Let's give an orientation to the tandem repeat by adding the string "5'-" (5' end) on the left and "-3'" (3' end) to the right.

```
motif = "ATTCG"

tandem_repeat = motif * 5

print(motif)
print(tandem_repeat, " has length", len(tandem_repeat))
m1 = "TCGAT"
m2 = "TCCT"

print("Is ", m1, " in ", tandem_repeat, " ? ", m1 in tandem_repeat )
print("Is ", m2, " in ", tandem_repeat, " ? ", m2 in tandem_repeat )
oriented_tr = "5\'-" + tandem_repeat + "-3\'"
print(oriented_tr)
```

```
ATTCG
ATTCGATTCGATTCGATTCGATTCG  has length 25
Is  TCGAT  in  ATTCGATTCGATTCGATTCGATTCG  ?  True
Is  TCCT  in  ATTCGATTCGATTCGATTCGATTCG  ?  False
5'-ATTCGATTCGATTCGATTCGATTCG-3'
```

Strings



Indexing and Slicing

Indexing starts from 0

str[i] : i+1-th character

str[S:E:step] slice string

Remember: S inclusive, E exclusive

```
S = "Luther College"

print(S) #print the whole string
print(S == S[:]) #a fancy way of making a copy of the original string
print(S[0]) #first character
print(S[3]) #fourth character
print(S[-1]) #last character
print(S[0:6]) #first six characters
print(S[-7:]) #final seven characters
print(S[0:len(S):2]) #every other character starting from the first
print(S[1:len(S):2]) #every other character starting from the second
```

```
Luther College
True
L
h
e
Luther
College
Lte olg
uhrClee
```

Strings

Methods

Result	Method	Meaning
str	str.upper()	Return the string in upper case
str	str.lower()	Return the string in lower case
str	str.strip(str)	Remove strings from the sides
str	str.lstrip(str)	Remove strings from the left
str	str.rstrip(str)	Remove strings from the right
str	str.replace(str, str)	Replace substrings
bool	str.startswith(str)	Check if the string starts with another
bool	str.endswith(str)	Check if the string ends with another
int	str.find(str)	Return the first position of a substring starting from the left
int	str.rfind(str)	Return the position of a substring starting from the right
int	str.count(str)	Count the number of occurrences of a substring

IMPORTANT NOTE Since Strings are immutable, every operation that changes the string actually produces a new *str* object having the modified string as value.

Strings

Example: Given the DNA sequence S = " aTATGCCCATatcgctAAATTGCTGCCATTACA ". Print its length (removing any blank spaces at either sides), the number of adenines, cytosines, guanines and thymines present. Is the sequence "ATCG" present in S? Print how many times the substring "TGCC" appears in S and all the corresponding indexes.

```
S = "   aTATGCCCATatcgctAAATTGCTGCCATTACA    "

print(S)
S = S.strip(" ")
print(S)

print(len(S))
tmpS = S.upper() #for simplicity to count only 4 different nucleotides
print("A count: ", tmpS.count("A"))
print("C count: ", tmpS.count("C"))
print("T count: ", tmpS.count("T"))
print("G count: ", tmpS.count("G"))
print("Is ATCG in ", tmpS, "? ", tmpS.find("ATCG") != -1) #or tmpS.count("ATCG") > 0
print("TGCC is present ", tmpS.count("TGCC"), " times in ", tmpS)
print("TGCC is present at pos ", tmpS.find("TGCC"))
print("TGCC is present at pos ", tmpS.rfind("TGCC")) #or tmpS.find("TGCC",4)
```

```
   aTATGCCCATatcgctAAATTGCTGCCATTACA
aTATGCCCATatcgctAAATTGCTGCCATTACA
33
A count:  10
C count:  9
T count:  10
G count:  4
Is ATCG in  ATATGCCCATATCGCTAAATTGCTGCCATTACA ?  True
TGCC is present  2  times in  ATATGCCCATATCGCTAAATTGCTGCCATTACA
TGCC is present at pos  3
TGCC is present at pos  23
```

Strings

Example: Since the genetic code is degenerate, there are many codons encoding for the same aminoacid. Consider Proline, it can be encoded by the following codons: CCU, CCA,CCG, CCC. Let's create a string proline and assign it to its possible codons one after the other.

```
"""
Wrong solution. We cannot directly replace the value of a string
"""

proline = "CCU"
print("Proline can be encoded by: ", proline)
proline[2]="A"
print(".. or by: ", proline)
```

Strings

Example: Since the genetic code is degenerate, there are many codons encoding for the same aminoacid. Consider Proline, it can be encoded by the following codons: CCU, CCA,CCG, CCC. Let's create a string proline and assign it to its possible codons one after the other.

Remember:

strings are immutable and do not support item assignment

```
"""
Wrong solution. We cannot directly replace the value of a string
"""

proline = "CCU"
print("Proline can be encoded by: ", proline)
proline[2]="A"
print(".. or by: ", proline)
```

```
Proline can be encoded by:  CCU
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-9750dcfa1cbd> in <module>()
      5 proline = "CCU"
      6 print("Proline can be encoded by: ", proline)
----> 7 proline[2]="A"
      8 print(".. or by: ", proline)
      9

TypeError: 'str' object does not support item assignment
```

Strings

Example: Since the genetic code is degenerate, there are many codons encoding for the same aminoacid. Consider Proline, it can be encoded by the following codons: CCU, CCA,CCG, CCC. Let's create a string proline and assign it to its possible codons one after the other.

Remember:

strings are immutable and do not support item assignment

```
"""
Correct solution. Using str.replace
"""
proline = "CCU"
print("Proline can be encoded by: ", proline)
proline = proline.replace("U","A")
print(".. or by: ", proline)
proline = proline.replace("A","G")
print(".. or by: ", proline)
proline = proline.replace("G","C")
print(".. or by: ", proline)
```

```
Proline can be encoded by:  CCU
.. or by:  CCA
.. or by:  CCG
.. or by:  CCC
```

Strings

Example: Since the genetic code is degenerate, there are many codons encoding for the same aminoacid. Consider Proline, it can be encoded by the following codons: CCU, CCA,CCG, CCC. Let's create a string proline and assign it to its possible codons one after the other.

Remember:

strings are immutable and do not support item assignment

```
"""
Another correct solution. Using string slicing and catenation.
"""
"""
Correct solution. Using str.replace
"""
proline = "CCU"
print("Proline can be encoded by: ", proline)
proline = proline[:-1]+"A" #equal to proline[0:-1] or proline[0:2]
print(".. or by: ", proline)
proline = proline[:-1]+"G"
print(".. or by: ", proline)
proline = proline[:-1]+"C"
print(".. or by: ", proline)
```

```
Proline can be encoded by:  CCU
.. or by:  CCA
.. or by:  CCG
.. or by:  CCC
```

Questions ?

https://qcbsciprolab.readthedocs.io/en/latest/practical2.html

Go quickly through the text and do the exercises at the end

Exercises

1. An exon of a gene starts from position 12030 on a genome and ends at position 12174. Does an A/T SNP present at position 12111 affect this exon? And what about a SNP present at position 12188?

Show/Hide Solution

2. SNP FB_AFFY_0000024 of the Apple 480K SNP chip has 5' flanking region (i.e. the forward probe) CATTATTTTCACTTGGGTCGAGGCCAGATTCCATC and 3' flanking region (i.e. reverse probe) GGATTGCCCGAAATCAGAGAAAAGTCG. The SNP is a G/A transversion. Answer the following questions:
   1. What is the length of the 5' flanking region? And that of the 3' flanking region?
   2. The IUPAC code of the G/A transversion is R. What is the sequence of the whole region using the [G/A] notation for the SNP (hint: concatenate in a new string called *region*) and the iupac notation R (region_iupac)?
   3. Retrive and print only the SNP from *region* and *iupac_region*

Show/Hide Solution

3. Compute the melting temperature T_m of the primer with sequence "TTAGCACACGTGAGCCAATGGAGCAAACGGGTAATT". The melting temperature T_m (in degrees Celtius) can be computed as: $T_m = 64.9 + 0.41(GC - 16.4)/N$, where GC is the total number of G and C in the primer and N is its length.

Show/Hide Solution