Introducció a Python: Sessió2:Dades(1) i control de flux

1. Tipus de dades

Una variable o constant pot contenir valors de diversos tipus.

1. **Booleà** (Cert / Fals):

```
vertader = True
fals = False
```

2. Els tipus **numèrics** són:

2.1. **Nombre enter (int)**:

que es pot expressar com a decimal,

```
edad = 35
```

com a nombre octal (comença amb 0) i

```
edat = 043
```

com a nombre hexadecimal (comença amb 0x)

```
edat = 0x23
```

2.2 **Nombre real (float)**Conté un punt decimal. Pot tenir mantisa i exponent

```
preu = 7435.28
preu2 = .01e+3
```

2.3. **Nombres Complexos**

```
num_comp1 = 4.53e-7j # j indica el component imaginàri.
```

3. Els tipus de dades anomenats **seqüències** permeten emmagatzemar una col·leccions de dades ordenades.

3.1. **Cadena de text** (**string**).

Són col·leccions ordenadas de carácters. Tenen la característica de ser **immutables**: quan es realitzen operacions amb strings, no es modifica l'string original sino que el resultat és un nou string. Es representen amb cometes, simples o dobles, dins de les quals hi ha els caracters que conté.

```
cadena1 = "Hola"
cadena_multilinia = """
Això és una cadena
de dues línies
"""
```

3.2. **Llista**.

Una llista és una col·lecció ordenada d'elements que poden ser de qualsevol tipus. Les llistes són **mutables**: es poden modificar. Les llistes es representen amb claudators **[]**, dintre dels quals s'escriuen els elements que conté.

Les llistes és l'estructura de Python que més s'assembla als vectors o arrays d'altres llenguatge. Si els elements de la llista són altres llistres, llavors tenim matrius ...

```
llista_buida = []
llista1 = ['cadena de text', 15, 2.8, 'altres dades', 25]
```

Els seus elements es poden **accedir** per **índex**, començant per 0. Si s'indica un í**ndex negatiu**, es refereix als elements començant pel final: l'índex -1 és el de l'últim element, l'index -2 és el del penúltim element etc. També es pot indicar un **interval**. En aquest cas, l'extrem dret no queda inclòs. Un interval sense límit inferior, indica "des del principi" i sense límit superior, "fins al final".

```
llista1[1] # retorna: 15
llista1[1:4] # retorna: [15, 2.8, 'altres dades']
llista1[-2] # retorna: 'altres dades'
```

Els elements de les llistes es poden modificar:

```
llista1[2] = 3.8 # el tercer element ara és 3.8
```

També es poden crear amb la funció predefinida **list()**, que rep com paràmetre una seqüència i retorna una llista que conté els elements de la seqüència.

```
list("Python") # retorna ['P', 'y', 't', 'h', 'o', 'n']
```

3.3. **Tupla**.

És una col·lecció ordenada d'elements que poden ser de qualsevol tipus. Les llistes són **immutables**: no es poden modificar. Les operacions de modificació generen una tupla nova i la original queda igual. Les tuple es representen amb parèntesis **()**, dintre dels quals s'escriuen els elements que conté.

```
tupla_buida = ()
tupla1 = ('cadena de text', 15, 2.8, 'altres dades', 25)
```

Els seus elements es poden **accedir** per **índex**, igual que les llistes:

```
print mi_tupla[1] # retorna: 15
```

```
tupla1[1:4] # retorna: (15, 2.8, 'altres dades')
tupla1[3:] # retorna: ('altres dades', 25)
tupla1[:2] # retorna: ('cadena de text', 15)
```

```
tupla1[-1] # retorna: 25
tupla1[-2] # retorna: 'altres dades'
```

També es poden crear amb la funció predefinida **tuple()**, que es comporta de manera anàloga a **list()**.

4. Els tipus de dades anomenats **sets** i **mappings (conjunts** i **correspondències)** emmagatzemen dades no ordenades.

4.1. **Diccionari**.

És una col·lecció **no ordenada** de parelles **clau:valor**. Els diccionaris són **mutables**: es poden modificar. Es representen amb parèntesis claus **{}**, que contenen les parelles clau:valor, separades per comes.
Les claus i els valors poden ser objectes de qualsevol tipus. La única restricció és que **no hi pot haver claus repetides**. Si s'afegeixen parells amb la mateixa clau, l'últim afegit és el que queda.

Es pot accedir als elements d'un diccionari indicant la clau:

```
diccionari1 = {'clau_1': valor_1, 'clau_2': valor_2, \
'clau_7': valor_7}

diccionari1['clau_2'] # retorna: valor_2
```

Els elements també es poden modificar

```
diccionari1['clau_1'] = 'Nou Valor'
```

i eliminar

```
del(diccionari1['clau_2'])
```

La creació es pot fer també amb la funció predefinida **dict()**, que rep com a paràmetre una llista de llistes de dos elements.

La sentència

```
dict ([['Jordan' , 23], ['Messi',: 10]])
```

genera el diccionari

```
{'Jordan' : 23, 'Messi' : 10}
```

3.3. **Conjunt (set).**

Es creen amb la funció set(), que rep una seqüència com a paràmetre.
Els conjunts contenen elements **no ordenats** i **no repetits**.

2. Sentències i control de flux

2.1 Assignació

L'operador d'assignació en Python és el **=**.

```
a = 'string'
```

També es disposa de l'**assignació abreviada** en què es realitza també una operación aritmetica. Els operadors són +=, -=, *= etc,

Es permet l'**assignació mùltiple**

```
a, b, c = 'string', 15, True
```

```
>>> print a
string
>>> print b
15
>>> print c
True
```

2.2 Estructura condicional

Per prendre decisions i trencar l'estructura linial d'un programa, s'utilitza la instrucció *if* .

```
1  a=2
2  b=3
3  if a==b:
4    #Si a i b són iguals (noteu el doble ==)
5    print("són iguals")
6  else:
7    print("són diferents")
```

El contingut del if queda "dintre" perquè l'escribim indentat.
Després de la condició del if i del else és obligatori escriure **:**. Aquests dos punts indiquen l'inici de bloc.

La clàusula else es opcional

```
1  a=input("escriu Hola al revés:")
2  if a=="aloH":
3     print("éB tloM")
```

Quan utilitzem estructures niades amb *if … else*, es pot utilitzar la instrucció *elif* (contracció d'else i if).

Els següents fragments de codi són equivalents, però la instrucció *elif* produeix un codi més compacte:

```
#Versió 1
edat_usuari=int(input("edat?"))
print("Tens ")
if edat_usuari==1:
    print("un any")
else:
     if edat_usuari==2:
         print("dos anys")
     else:
        if edat_usuari==3:
            print("tres anys")
        else:
            if edat_usuari==4:
              print("quatre anys")
            else:
                    print("més de 4 anys")
```

```
#Versió 2
edat_usuari=int(input("edat?"))
print("Tens ")
if edat_usuari==1:
  print("un any")
elif edat_usuari==2:
  print("dos anys")
elif edat_usuari==3:
  print("tres anys")
elif edat_usuari==4:
  print("quatre anys")
else:
  print("més de 4 anys")
```

En les condicions es poden utilitzar els operadors relacionals

```
if a==b:#Iguals
if a!=b:#Diferents
if a>b:#a major que b
if a>=b:#a major o igual que b
if a<b:#a menor que b
if a<=b:#a menor o igual que b
```

i els operadors lògics:

and (y): Retorna true si totes les condicions són veritat, sinò, retorna false.
or (ó): Retorna true si alguna condició és veritat, sinò, retorna false.
not (no): Retorna el contrari de la condició.

Exemple amb **and**

```
a=int(input("Quants anys tens?"))
if (a>=16) and (a<=67):
    print("Pots treballar.")
else:
    print("No pots treballar.")
```

Exemple amb **or**

```
print("Entra tres números majors que 10")
a=int(input("Primer número?"))
b=int(input("Segon número?"))
c=int(input("Tercer número?"))
if (a<10) or (b<10) or (c<10):
    print("            T'has equivocat en, com a mínim, un número ")
else:
    print("Números correctes")
```

Exemple amb **not**

```
print("Entra dos números positius?")
a=int(input("Primer número?"))
b=int(input("Segon número?"))
if not(a>0 and b>0):
    print("          T'has equivocat en, com a mínim, un número ")
else:
    print("Números correctes")
```

2.3 Estructures iteratives: bucle while

Un bucle **while** és una instrucció que permet repetir l'execució d'un grup d'instruccions mentre es compleixi una condició booleana.
La variable o les variables que apareguin en la condició se solen anomenar variables de control. Les variables de control han de definir-se **abans** del bucle **while**.

L'estructura d'un bucle while és:

while econdició**:**
sentències

El següent programa escriu els nombres de l'1 al 10:

```
i = 1
while i <= 10:
    print i
    i += 1
```

Programa que demana un número positiu i, fins que no l'aconsegueix, el torna a demanar.

```
print "Escriu un número positiu:"
numero = int(raw_input())
while numero < 0:
    print("Aquest número és negatiu, entra'n un de positiu")
    numero = int(raw_input())
print "Gràcies"
```

Programa que cerca un número entre 1 i 10000 divisible entre 15, 16 i 26

```
numero = 1
sortir = False
while sortir==False:
    if numero%15==0 and numero%16==0 and numero%26==0:
        sortir=True
    else:
        numero=numero+1
        if numero>10000:
            sortir=True
print "Un numero divisible entre 15, 16 i 26:"
if numero>10000:
    print "No l'he trobat"
else:
    print numero
```

L'execució del bucle while es pot canviar amb les sentències break (surt del bucle) i continue (pasa directament a la següent iteración)

2.4 Estructures iteratives: bucle for

En Python, el bucle for té l'estructura:

for element **in** expressió_iterable**:**
sentències

'expressió_iterable' és qualsevol objecte que es pot recórrer, és a dir, sobre el qual es pot iterar (per mitjà d'un objecte iterador generat implícitament).

En aquest exemple, iterem sobre una llista:

```
for i in ["/","-","|","\\","|"]:
    print "%s\r" % i,
```

Observeu que i representa l'element de l'expressió iterable que es considera en cada volta del bucle.

Un iterable molt utilitzat en els bucles for és la seqüència **range**.
Un range és una progressió aritmètica generada per la funció range(). Els paràmetres de la funció range són:
range([valor_inicial], valor_final, [increment])

El valor inicial i l'increment són opcionals. Si no s'indica increment, es pren 1. Si no s'indica valor incial ni increment, es pren 0 i 1 respectivament. El valor final **NO** s'inclou en els valors generats.

Exemples:
range(1, 10, 2) genera 1 3 5 7 9

range(22. 10, -2) genera 22 20 18 16 14 12
range(4) genera 0 1 2 3

Exemple de bucle for utilitzant un range com a element iterable:

```
for i in range(1, 10, 2)
    print i
```

L'execució del bucle for es pot canviar amb les sentències break (surt del bucle) i continue (pasa directament a la següent iteración)

3. Funcions

Parlant amb poques paraules i sense gaire formalitat, les **funcions** són trossos de codi reaprofitable. La seva sintaxi és

def nom_de_la_funció (paràmetres_si_n'hi_ha)**:**
cos de la funció

Exemple de definició de funció i de dues crides diferents:

```
def imprimeix_taula(numero): #definim una funció
    '''imprimeix_taula(numero)
    què fa: imprimeix la taula de multiplicar del número entrat
    parámetres: numero -> enter
    retorna: res
    '''
    for i in range(10):
        print "%d*%d=%d" % (numero,i,numero*i)

imprimeix_taula(3)
print "\n"
imprimeix_taula(9)
```

Les línies 2 a 5 són comentaris que descriuen el que fa la funció.

Les funcions poden retornar dades.

Exemple de definició de funció i crida d'una funció amb retorn:

```
def es_major_d_edat(edat): #definim una funció
    ''' es_major_d_edat(edat)
    què fa: retorna True o Fals si és major d'edat
    parámetres: edat ->enter
    retorna: True o Fals
    '''
    if edat>=18:
        return True
    else:
        return False


edat=int(input("edat?"))
if es_major_d_edat(edat):
    print "Tens edat penal"
else:
    print "No pots conduir un cotxe"
```

En la capçalera de la funció no s'indica si la funció retorna cap valor ni, si ho fa, el tipus del retorn.

Un altre exemple:

```
def retorna_area_circumferencia(radi):
    ''' retorna_area_circumferencia(radi)
    què fa: retorna el valor de l'àrea d'una circumferència
    parámetres: radi ->enter
    retorna: L'àrea -> float d'una circumferència
    '''
    pi=3.14
    return pi*radi*radi


radi=3.0
area=retorna_area_circumferencia(radi)
print "L'àrea d'una circumferència de radi %.2f és de %.2f" % (radi,area)
```

4. Activitats de la Sessió 2

1. Funcions.

Crear un fitxer d'script amb el següent contingut:

```
def cheese_and_crackers(cheese_count, boxes_of_crackers):
    print "You have %d cheeses!" % cheese_count
    print "You have %d boxes of crackers!" % boxes_of_crackers
    print "Man that's enough for a party!"
    print "Get a blanket.\n"


print "We can just give the function numbers directly:"
cheese_and_crackers(20, 30)


print "OR, we can use variables from our script:"
amount_of_cheese = 10
amount_of_crackers = 50

cheese_and_crackers(amount_of_cheese, amount_of_crackers)


print "We can even do math inside too:"
cheese_and_crackers(10 + 20, 5 + 6)


print "And we can combine the two, variables and math:"
cheese_and_crackers(amount_of_cheese + 100, amount_of_crackers + 1000)
```

Executar-lo. Explicar d'on surten en cada cas els valors que es passen a la funció.
Programar una variació del programa en què es demani a l'usuari els nombres de formatges i aperitius. Utilitzar la funció int() per transformar a enter el valor retornat per raw_input().

2. Funcions amb retorn.

Crear un fitxer d'script amb el següent contingut:

```
def add(a, b):
    print "ADDING %d + %d" % (a, b)
    return a + b

def subtract(a, b):
    print "SUBTRACTING %d - %d" % (a, b)
    return a - b

def multiply(a, b):
    print "MULTIPLYING %d * %d" % (a, b)
    return a * b

def divide(a, b):
    print "DIVIDING %d / %d" % (a, b)
    return a / b


print "Let's do some math with just functions!"

age = add(30, 5)
height = subtract(78, 4)
weight = multiply(90, 2)
iq = divide(100, 2)

print "Age: %d, Height: %d, Weight: %d, IQ: %d" % (age, height, weight, iq)


# A puzzle for the extra credit, type it in anyway.
print "Here is a puzzle."

what = add(age, subtract(height, multiply(weight, divide(iq, 2))))

print "That becomes: ", what, "Can you do it by hand?"
```

Executar-lo. Explicar com es calcula el valor que acaba en la variable *what*. Programar una variació del programa en què es demani a l'usuari els valors (nombres reals) per fer les operacions. Utilitzar la funció float() per transformar en nombre real el valor retornat per raw_input().

3. Funcions amb retorn.

Crear un fitxer d'script amb el següent contingut:

```
print "Let's practice everything."
print 'You\'d need to know \'bout escapes with \\ that do \n newlines and \t tabs.'

poem = """
\tThe lovely world
with logic so firmly planted
cannot discern \n the needs of love
nor comprehend passion from intuition
and requires an explanation
\n\t\twhere there is none.
"""

print "--------------"
print poem
print "--------------"


five = 10 - 2 + 3 - 6
print "This should be five: %s" % five

def secret_formula(started):
    jelly_beans = started * 500
    jars = jelly_beans / 1000
    crates = jars / 100
    return jelly_beans, jars, crates


start_point = 10000
beans, jars, crates = secret_formula(start_point)

print "With a starting point of: %d" % start_point
print "We'd have %d beans, %d jars, and %d crates." % (beans, jars, crates)

start_point = start_point / 10

print "We can also do that this way:"
print "We'd have %d beans, %d jars, and %d crates." % secret_formula(start_point)
```

Executar-lo. Explicar com funciona el retorn de múltiples valors.
Explicar quantes variables anomenades jars i crates hi ha i la seva visibilitat.

4. Funcions amb retorn que criden altres funcions.

Crear un fitxer d'script anomenat **ex25.py** amb el següent contingut:

```
def break_words(stuff):
    """This function will break up words for us."""
    words = stuff.split(' ')
    return words

def sort_words(words):
    """Sorts the words."""
    return sorted(words)

def print_first_word(words):
    """Prints the first word after popping it off."""
    word = words.pop(0)
    print word

def print_last_word(words):
    """Prints the last word after popping it off."""
    word = words.pop(-1)
    print word

def sort_sentence(sentence):
    """Takes in a full sentence and returns the sorted words."""
    words = break_words(sentence)
    return sort_words(words)

def print_first_and_last(sentence):
    """Prints the first and last words of the sentence."""
    words = break_words(sentence)
    print_first_word(words)
    print_last_word(words)

def print_first_and_last_sorted(sentence):
    """Sorts the words then prints the first and last one."""
    words = sort_sentence(sentence)
    print_first_word(words)
    print_last_word(words)
```

Provisionalment, afegir codi que cridi a les funcions, i executar-lo, per assegurar que no hi ha cap error sintàctic. Els paràmetres anomenats *statement* han de rebre una frase com a argument.
Quan estigui tot correcte, deixar només les línies que es mostren a dalt.

Fer un esquema, gràficament o amb text, que mostri les cadenes de crides de funcions.

Comprendre i explicar què fa cadascuna de les funcions, les codificades a l'script i les predefinides sorted, pop i split.

5. Funcions amb retorn que criden altres funcions.

Utilitzar l'script creat en l'exercici anterior i executar-lo interactivament, des de l'intèrpret de Pyton.
La sessió amb l'intèrpret hauria de ser similar a

```
Python 2.7.11 (default, May 25 2016, 05:27:56)
[GCC 4.2.1 Compatible Apple LLVM 7.3.0 (clang-703.0.29)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import ex25
>>> sentence = "All good things come to those who wait."
>>> words = ex25.break_words(sentence)
>>> words
['All', 'good', 'things', 'come', 'to', 'those', 'who', 'wait.']
>>> sorted_words = ex25.sort_words(words)
>>> sorted_words
['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
>>> ex25.print_first_word(words)
All
>>> ex25.print_last_word(words)
wait.
>>> words
['good', 'things', 'come', 'to', 'those', 'who']
>>> ex25.print_first_word(sorted_words)
All
>>> ex25.print_last_word(sorted_words)
who
>>> sorted_words
['come', 'good', 'things', 'those', 'to', 'wait.']
>>> sorted_words = ex25.sort_sentence(sentence)
>>> sorted_words
['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
>>> ex25.print_first_and_last(sentence)
All
wait.
>>> ex25.print_first_and_last_sorted(sentence)
All
who
```

Explicar l'efecte de cada ordre que es dóna a l'intèrpret.
Teclejar help(ex25) i help(ex25.break_words). Explicar el que s'obté.

6. Control de flux.

Crear un script amb el següent contingut.

```
print "You enter a dark room with two doors.  Do you go through door #1 or door #2?"

door = raw_input("> ")

if door == "1":
    print "There's a giant bear here eating a cheese cake.  What do you do?"
    print "1. Take the cake."
    print "2. Scream at the bear."

    bear = raw_input("> ")

    if bear == "1":
        print "The bear eats your face off.  Good job!"
    elif bear == "2":
        print "The bear eats your legs off.  Good job!"
    else:
elif door == "2":
    print "You stare into the endless abyss at Cthulhu's retina."
    print "1. Blueberries."
    print "2. Yellow jacket clothespins."
    print "3. Understanding revolvers yelling melodies."

    insanity = raw_input("> ")

    if insanity == "1" or insanity == "2":
        print "Your body survives powered by a mind of jello.  Good job!"
    else:
        print "The insanity rots your eyes into a pool of muck.  Good job!"

else:
    print "You stumble around and fall on a knife and die.  Good job!"
```

Executar i comprendre el programa.
Modificar el programa de manera que el joc es repeteixi indefinidament fins que l'usuari digui que no vol continuar.

7. Bucle for.

Crear un script amb el següent contingut.

```
1   the_count = [1, 2, 3, 4, 5]
2   fruits = ['apples', 'oranges', 'pears', 'apricots']
3   change = [1, 'pennies', 2, 'dimes', 3, 'quarters']
4
5   # this first kind of for-loop goes through a list
6   for number in the_count:
7       print "This is count %d" % number
8
9   # same as above
10  for fruit in fruits:
11      print "A fruit of type: %s" % fruit
12
13  # also we can go through mixed lists too
14  # notice we have to use %r since we don't know what's in it
15  for i in change:
16      print "I got %r" % i
17
18  # we can also build lists, first start with an empty one
19  elements = []
20
21  # then use the range function to do 0 to 5 counts
22  for i in range(0, 6):
23      print "Adding %d to the list." % i
24      # append is a function that lists understand
25      elements.append(i)
26
27  # now we can print them out too
28  for i in elements:
29      print "Element was: %d" % i
```

Executar-lo. Explicar el funcionament de cadascun dels bucles for que hi ha. Explicar l'efecte de la funció (en realitat, és un mètode) append.

8. Bucle while.

Crear un script amb el següent contingut.

```
i = 0
numbers = []

while i < 6:
    print "At the top i is %d" % i
    numbers.append(i)

    i = i + 1
    print "Numbers now: ", numbers
    print "At the bottom i is %d" % i


print "The numbers: "

for num in numbers:
    print num
```

Executar-lo. Explicar el funcionament de cadascun dels bucles que hi ha.