UVIC
UNIVERSITAT
DE VIC

Introducció al PYTHON:

Tipus de Dades i

Estructures de Control

Introducció a la programació

Professorat: **Cristina Borralleras**

Català

Autoria: M. Dolors Anton i Solà

No es pot copiar sense permís de l'autor/a.

Drets reservats © 2012
Universitat de Vic

Sagrada Família, 7
08500 Vic (Barcelona)

Índex

Introducció

L'objectiu d'aquesta unitat és el d'introduir a l'estudiant al llenguatge de programació PYTHON.

Es vol que l'estudiant assoleixi els conceptes bàsics i necessaris per poder escriure un programa.

Avui en dia independentment de la disciplina en que es treballi, cal tenir coneixements de programació, doncs cada vegada s'automatitzen més les tasques, i que millor que davant d'una tasca a resoldre, que sigui prou complexa, saber escriure un programa i tenir la solució automatitzada.

Objectius

1. Saber que és PYTHON.
2. Conèixer els diferents tipus de dades.
3. Conèixer les estructures de control bàsiques de programació.
4. Tenir la capacitat d'escriure un programa en PYTHON que manipuli tot tipus de dades elementals.

Continguts

Unitat Didàctica 1. **Introducció al PYTHON: tipus de dades i estructures de control**

1. INTRODUCCIÓ al PYTHON

L'objectiu principal d'aquesta assignatura és que l'estudiant tingui facilitat per desenvolupar programes en PYTHON, i que a més amb la base de programació que adquireixi li sigui fàcil programar en qualsevol altre llenguatge.

Que s'entén per programa?

> *Un programa és un conjunt d'instruccions escrites en un llenguatge de programació que la CPU de l'ordinador va executant per tal de realitzar una tasca determinada.*

De llenguatges de programació n'hi ha molts i de diferents tipus, en aquesta assignatura s'ha escollit el PYTHON per ser un dels llenguatges més usant dins l'àrea de les biociències. Cal dir que tot llenguatge de programació té una gramàtica associada amb unes regles sintàctiques molt estrictes, és a dir que el mínim error sintàctic o gramatical en una instrucció d'un programa pot fer que doni resultats erronis.

Un consell a donar, que es veurà que l'anem seguint durant el curs, és que sempre que es vulgui generar un programa cal seguir uns passos: primer entendre el que ens demanen, després planejar com solucionar-lo i finalment escriure'l en el llenguatge de programació corresponent.

Què és PYTHON?

- Llenguatge de programació creat per Guido van Rosum.
- Llenguatge de programació d'alt nivell, això vol dir que necessita un altre programari, compilador o intèrpret que entengui les instruccions i les tradueixi a llenguatge màquina. El compilador és aquell programari que agafa les instruccions escrites en un llenguatge d'alt nivell, les tradueix a llenguatge màquina i genera un fitxer executable que pot ser executar des de qualsevol lloc. Un llenguatge interpretat necessita sempre del programari de l' intèrpret per tal de poder ser executat.
- Llenguatge de programació interpretat: usa l' intèrpret per entendre cada ordre que se li dóna i executar-la.
- Multiplataforma: L' intèrpret de PYTHON està disponible en moltes plataformes com ara UNIX, WINDOWS, Mac OS,…
- Orientat a objectes: paradigma de la programació que permet acostar-se als aspectes més importants del món real.

- Sintaxi senzilla: semblant al llenguatge natural, per tant un bon llenguatge quan es vol aprendre a programar.

Com començar a usar el PYTHON?

Tal i com s'ha dit és un llenguatge interpretat, per tant s'ha de tenir primer de tot l' intèrpret engegat.

Un cop es té l' intèrpret ja engegat, apareix la següent finestra:

Es pot treballar de dues maneres:

- Escriure la línia de codi directament a l' intèrpret i ens dóna el resultat,

o bé,

- Escriure el conjunt de línies de codi d'un programa en un fitxer i després executar-lo.

El més normal és escollir al segona opció doncs un programa està compost per moltes instruccions, per tant amb un cop que s'escriguin n'hi haurà suficient, i després només caldrà donar l'ordre d'execució del fitxer cada vegada que es vulgui.

Per tal de conèixer bé la sintaxi, es creu que el millor és començar per un petit programa i anar-lo ampliant amb noves necessitats fins arribar a conèixer tota la sintaxi prevista dins l'assignatura.

Primer programa en PYTHON

Si es vol fer servir algun programa que es tingui en l'ordinador, com molt bé es sap, cal demanar primer la seva execució, a continuació donar la informació que es necessiti per poder-lo executar i després veure el resultat obtingut acabada la seva execució. Per tant hi ha un intercanvi d'informació entre l'usuari i l'ordinador.

Així doncs si es vol començar a programar en PYTHON cal saber primer de tot quines són les instruccions bàsiques d'entrada i sortida d'informació.

Com fer que es mostri un resultat per pantalla?

La instrucció és ben simple:

```
print("el que vulguem escriure")
```

Exemple:

```
>> print(3)
3
>> print("bon dia")
bon dia
```

Observant les dues instruccions es pot veure que si s'escriu un valor numèric no el posem entre cometes, en canvi si és un text si. Ja es comencen a veure les primeres restriccions de sintaxi.

Aquí en els exemples les instruccions estan escrites directament sobre l' intèrpret, si es volen escriure en un fitxer, es pot fer usant el gedit, com?

- Obrir el gedit
- Escriure la o les línies de codi que componen el programa
- Guardar-lo com nomprograma.py

Per fer ara la seva execució cal:

- Estant a terminal escriure PYTHON nomprograma.py, suposant que el fitxer està en el mateix directori on estem posicionats.

o bé

- Estant a terminal escriure PYTHON cami/nomprograma.py, indicant en el camí el directori on es troba el fitxer.

o bé

- Afegir una línia al principi del fitxer, just abans de les línies de codi, anomenada shebang, que serveix per indicar la ruta on es troba el PYTHON, independentment on s'executi el programa. El contingut d'aquesta línia és:

```
#!/usr/bin/env python
```

Amb aquesta opció simplement fent un doble clic sobre el fitxer ja 'executarà, sempre hi quan se li hagin assignat els privilegis d'execució. Per assignar els privilegis d'execució a un fitxer cal executar l'ordre:

```
chmod +x nomfitxer.extensió
```

El resultat del primer programa escrit dins un fitxer tenint en compte el shebang és:

```
#!/usr/bin/env python
print("bon dia")
print(3)
```

Si s'executa el programa no hi ha temps per veure el resultat, quan el normal és que interessi veure'l, per tant s'afegirà la instrucció:

```
input()
```

Aquesta instrucció el que fa és esperar que l'usuari premi la tecla INTRO o qualsevol altre tecla, per tant permet que es pugui veure el resultat.

Com llegir un valor entrat per teclat?

L' instrucció anterior permet llegir un valor entrat per teclat. Llavors si interessa poder manipular aquest valor cal que s'assigni a una variable. La instrucció que permet això és:

```
nomvariable = input()
```

Què s'entén per variable?

Espai de memòria destinat a guardar alguna dada.

Remarcar que de dades n'hi ha de molts tipus, dels que cal conèixer les seves propietats i les operacions que se'ls pot aplicar, per després poder-les manipular.

2. TIPUS DE DADES

PYTHON té 3 grups de dades:

- **Numèrics**

  Dins d'aquest grup es distingeixen 3 tipus:

  - Enters → positius, negatius i zero, sense decimals

    ```
    a=3
    ```

  - Reals → seran els números amb decimals. Remarcar que la coma decimal en PYTHON es correspon al punt.

    ```
    r=0.25
    ```

    Remarcar que es pot treballar amb notació científica afegint al número real la lletra e i l'exponent.

    ```
    r=1.2e-3     , que equival a  0.0012
    ```

  - Complexos → no es treballaran en l'assignatura.

Els operadors bàsics de PYTHON que permeten manipular tipus numèrics són:

OPERADOR	DESCRIPCIÓ	EXEMPLE
+	Suma	r = 3+2 r val 5
-	Resta	r= 3-2 r val 1
*	Producte	r= 3*2 r val 6
/	Divisió	r=6.5/2 r val 3.25
**	Exponencial	r= 3**4 r val 81
//	Divisió entera	r= 7//2 r val 3
%	Residu	r= 7%2 r val 1

Cal remarcar que amb Python 2.x, el tipus de numèric que s'obté després d'aplicar una operador sempre es correspon al tipus més general que intervé. Llavors si s'aplica la divisió entre dos números enters el resultat és un enter, encara que la divisió no sigui exacta:

```
r= 7/2
```
r val 3 - Amb Python 3.x r val 3.5

En aquets cas (Python 2.0) si és vol que es resultat sigui real cal que un dels dos s'escrigui com a real:

```
r= 7.0/2
```
r val 3.5

o bé,

```
r= float(7)/2
```
r val 3.5 , en aquest cas s'usa la funció float que converteix el dividend en real.

- **Cadenes (Strings)**

Es corresponen a text delimitat per cometes simples ('hola') o dobles ("hola").

Dins de les cometes hi pot haver qualsevol símbol que es pugui entrar per teclat.

Si s'escriu un text on interessa que es respectin els canvis de línia llavors s'ha de delimitar amb triple cometes.

Els operadors admesos per aquest tipus de dades són:

OPERADOR	DESCRIPCIÓ	EXEMPLE
+	Concatenació	a="hola" b= "adeu" c=a+b c val "holaadeu"
*	Repetició	a="hola" c=a*3 c val "holaholahola"

- **Booleans**

Aquest tipus de dades només pot tenir 2 valors: Cert (True) o Fals (False). Són tipus molt útils quan es treballa amb expressions condicionals.

Hi ha 2 tipus d'operadors que ens permeten treballar amb els valors booleans: els relacionals i els lògics. Tant els uns com els altres sempre tornen coma resultat un valor booleà.

LÒGICS

Els operands han de ser de tipus per poder aplicar aquests tipus d'operadors.

OPERADOR	DESCRIPCIÓ	EXEMPLE
and	Com a resultat dóna cert si i només si els dos operands prenen el valor de cert.	a=True; b=False; c=True r=a and b r val False r= a and c r val True
or	Com a resultat dóna cert si com a mínim un dels dos operands és cert	a=False; b=True; c=False r= a or b r val True r= a or c r val False
not	Coma resultat dóna el contrari al valor de l'operand	a=False; b= True r= not a r val True r=not b r val False

RELACIONALS

S'usen per fer comparacions entre valors.

OPERADOR	DESCRIPCIÓ	EXEMPLE
==	Dóna cert si els dos operands són iguals	7==2, el resultat és False
!=	Dóna cert si els dos operands són diferents	7!=2, el resultat és True
<	Dóna cert si l'operand de l'esquerra és més petit que el de la dreta.	7<2, el resultat és flase
>	Dóna cert si l'operand de l'esquerra és més gran que el de la dreta.	7>2, el resultat és True
<=	Dóna cert si l'operand de l'esquerra és més petit o igual que el de la dreta.	7<=7, el resultat és True 7<=2, el resultat és False
>=	Dóna cert si l'operand de l'esquerra és més gran o igual que el de la dreta.	7>=2, el resultat és True

3. ESTRUCTURES DE CONTROL

Recordant la definició de programa, " conjunt d'instruccions que executa un ordinador", s'entén que aquestes instruccions no s'executen simultàniament, sinó que s'executen una darrera l'altre, una instrucció s'executa immediatament després d'haver acabat l'anterior.

Una estructura de control permet indicar al programa l'ordre en que s'executen les accions.

Les 3 estructures de control que hi ha són:

Estructura seqüencial

Defineix una seqüència d'instruccions que s'executen una darrera l'altra. Aquesta estructura és l'usada per defecte.

Exemple:

Fer un programa que donats el radi i l'altura d'un cilindre en calculi la seva àrea i el seu volum:

```
print(" entra el radi:")
r=input()
r=float(r)
print("entra l'altura")
h=input()
h=float(h)
a=2*3.14*r**2 + 2*3.14*r*h
v=3.14*r**2*h
print(" àrea" + str(a))
print("volum" + str(v))
```

Cal fixar-se en que després de llegir el valor de radi i de l'altura s'ha escrit al funció float. Aquesta funció converteix el valor entrat a real. És necessari perquè la instrucció input() interpreta el que rep per teclat com una cadena, i això provoca que no s'hi puguin fer operacions aritmètiques.

Estructures condicionals

Aquest tipus d'estructura permet que s'executi un conjunt de instruccions o un altre depenent de condicions.

És molt normal que quan s'està escrivint un programa hi hagin moments que interessi que s'executin unes instruccions i en altres moments no. Aleshores per determinar la seva execució cal escriure una condició booleana. Aquí és on pren molta importància el tipus de dades booleà i els operadors relacionals i lògics associats al tipus.

La forma més simple d'estructura condicional és:

```
if  condició :
      instrucció1
      instrucció 2
      instrucció 3
            :
            :
            :
```

Si es compleix la condició (pren el valor de cert (True)) llavors s'executaran les instruccions.

IMPORTANT: el resultat que s'obté després d'avaluar la condició ha de ser sempre un valor booleà.

Observant la sintaxi es veu que les instruccions es comencen a escriure just després de la línia de **l'if** i indentades (mitjançant el tabulador estan mogudes cap a la dreta).

La INDENTACIÓ és molt important en PYTHON doncs permet marcar el conjunt de instruccions que s'han d'executar quan es compleix la condició. Quan s'escriu una instrucció no alineada com les anteriors l' intèrpret entén que ja és una instrucció que queda fora de la condició.

Exemple:

```
a= input()
b= input()
if a>b:
      print("ja tenim el primer número més gran")
      print(" moltes gràcies")
print(str(a)+ "   "+ str(b))
```

ULL!, si el valor que es dóna per tecla és numèric i interessa tractar-lo com a numèric també es pot escriure la instrucció input() enlloc de la input. Aquesta serveix per llegir valors numèrics. (Això només és vàlid amb PYTHON de versions inferiors a la 3.

En aquest exemple només s'escriurà:

ja tenim el primer número més gran
moltes gràcies

en cas de que a sigui més gran que b, però tant si es compleix com no la condició s'escriurà el valor de a i de b.

Si ara ens proposen el següent exercici:

Es sap que el llindar de glucosa en les persones és 120, llavors es demana un programa que permeti entrar una mesura feta a una persona i retorni el missatge de anar al metge en cas de superar el llindar.

El programa que satisfaci el requeriment és:

```
print(" entra la mesura feta de glucosa:")
g=input()
if  g>120:
     print(" ves a veure el metge")
     input()
```

Modificant l'enunciat anterior:

Es sap que el llindar superior de glucosa en les persones és 120 i l'inferior és 80 llavors es demana un programa que permeti entrar una mesura feta a una persona i retorni el missatge d'anar al metge en estar per sobre o sota respectivament dels llindars.

```
print(" entra la mesura feta de glucosa:")
g=input()
g=float(g)
if  (g>120) or (g<80):
     print(" ves a veure el metge")
     input()
```

Aquí en la condició ens hi apareix un operador lògic

Després de veure aquest exemples es comprova que només es duen a terme tasques si és compleix la condició. Ens podem trobar però amb casos on s'hagi de realitzar una tasca o una altra depenent de si és compleix o no la condició. Llavors la nova estructura de condicional que es necessita és:

```
if condicio:
      Instrucció1
      Instrucció2
      Instrucció3
      :
else:
      Instrucció4
      Instrucció5
      Instrucció6
      :
```

Si es compleix la condició realitzarà :
Instrucció 2
Instrucció 3
:
Si no es compleix, realitzarà:
Instrucció4
Instrucció 5
Instrucció 6
:

Tornant a l'exemple anterior:

Es sap que el llindar superior de glucosa en les persones és 120, llavors es demana un programa que permeti entrar una mesura feta a una persona i retorni el missatge de anar al metge en cas d'estar per sobre del llindar o bé el de tens un bon nivell de glucosa si no el sobrepassa.

```
print(" entra la mesura feta de glucosa:")
g=input()
g=float(g)
if  (g>120) :
     print(" ves a veure el metge")

else:
     print("tens un bon nivell de glucosa")
```

Fixar-se que la instrucció input() ja està alineada a la sentencia if, perquè s'executa fora de la condició, doncs independentment del missatge a donar s'executa aquesta sentencia per tal de poder veure el resultat.

L'última estructura condicional que cal conèixer és:

```
if condicio1:
     instrucció1
     instrucció 2
     instrucció 3
               :
elif condicio2:
     instrucció4
     instrucció 5
     instrucció 6
          :
elif condicio3:
     instrucció7
     instrucció 8
          :
:
:
else:
     instrucció_i
     instrucció _j
          :
```

Aquesta estructura avalua la condició1, i si es compleix executa les instruccions associades, en cas de no complir-se llavors avalua la condició2, que si es compleix executa les instruccions associades i en cas de no complir-se avalua la condicio3.., i així successivament fins a trobar l'else que té associades totes les instruccions a realitzar en cas de que no es compleixin cap de les condicions.

Tornant a l'exemple anterior i ampliant els requeriments:

Es sap que el llindar superior de glucosa en les persones és 120 i l'inferior és 80 llavors es demana un programa que permeti entrar una mesura feta a una persona i retorni el missatge de anar al metge en cas d'estar per sobre del llindar superior, o bé pren sucre aj en cas d'estar per sota del llindar inferior, o bé tens bon nivell de sucre si no es compleix cap dels casos anteriors.

```
print(" entra la mesura feta de glucosa:")
g=input()
if  (g>120) :
      print(" ves a veure el metge")

elif (g<80):
      print(" pren sucre")
else:
      print("tens un bon nivell de glucosa")
```

En aquest programa s'avalua si s'està per sobre el llindar, amb la condició g>120, enviant el missatge de ves a veure el metge, en cas de no estar-hi es mira si està per sota, amb la condició g<80, per tal d'enviar el missatge pren sucre i sinó es mostra tens un bon nivell de sucre.

Estructures iteratives

En les estructures condicionals s'executa un tros o un altre de codi depenen de condicions, però de vegades ens pot interessar que un fragment de codi s'executi un cert número de vegades, mentre es compleixi una condició. Aleshores per poder expressar-ho es necessiten les estructures iteratives.

Se'n veuran 2: WHILE i FOR..IN

WHILE

La seva sintaxi és:

```
while condicio:
      instruccio1
      instruccio2
      instruccio3
            :
```

El fragment d'instruccions que estan indentades s'executaran mentre es compleixi la condició.

La manera de treballar amb el while és:

1. S'avalua la condició
2. Si es compleix:
    a. S'executa el fragment de codi indentat
    b. Torna al punt 1

   Si no es compleix, es surt del while.

Modificant l'enunciat seguit fins ara:

Donada una sèrie de mesures fetes a diferents persones, entrada per teclat , acabada amb el valor 0 per indicar la finalització de mesures, comptar quantes persones tenen hipoglucèmia.

Pensant la solució:

1. Cal demanar una mesura.

2. Cal comprovar si una persona té hipoglucèmia. La té si està per sota del llindar inferior (<80). Aleshores aquí es necessita una estructura condicional per saber si és hipoglucèmica.

2. Es necessita algun lloc on poder anar guardant quantes persones tenen hipoglucèmia. Caldrà una variable que es vagi incrementant cada vegada que es trobi una persona amb hipoglucèmia. IMPORTANT: quan s'usa una variable que interessa que faci la funció de comptador cal que abans de començar a fer res s'inicialitzi a 0.

3. Un cop tractada la primera mesura, cal demanar una altra, i tornar al punt 2 si no és 0.

4.Quan la mesura llegida ja sigui 0 cal mostrar la variable que conté quantes persones són hipoglucèmiques.

Escrita en PYTHON:

```
compt=0
print(" entra la mesura feta de glucosa:")
g=input()
g=float(g)
while g!=0:
      if g<80:
            compt=compt=1
      print(" entra la mesura feta de glucosa:")
      g=input()
      g=float(g)
print(compt)
```

La variable g conté el valor de la glucosa entrat, que va canviant cada vegada que s'executa la instrucció input(), per tant la condició del while va controlant el contingut de g de tal manera que quan valgui 0 sabrà que s'ha acabat la seqüència de mesures.

Es posa a continuació un programa i fent el seguiment es descobrirà que fa:

```
compt=0
print("entra valor")
num=input()
while num!=0:
      if num%2==0:
            compt=compt+1
      print(" entra un altre valor")
      num=input()
print(compt)
```

1.Dóna el valor 0 a la variable num → inicialitza la variable

2. Demana que s'entri un valor i es guarda a la variable num. (Suposem que s'entra el 3)

3. S'avalua la condició del while, com que 3 és diferent de 0 entra dins del bucle.

4.S'avalua la condició de l'if. Aquesta condició retorna cert si el residu de dividir el número entrat entre 2 dóna 0, és a dir si es correspon a un número parell. Si ho és s'incrementa el comptador en 1.

5.Torna a demanar un nou valor i el torna a guardar a num, això suposa que el valor 3 que tenia fins ara queda perdut. Fins aquí totes les instruccions dins del while, per tant es tornarà a preguntar per la condició del while. Si es compleix es tornarà a fer el punt 3, 4 i 5, i sinó es compleix deixarà fer les instruccions de dins el while i continuarà realitzant les instruccions de després d'aquest, que en l'exemple es correspon a la mostrar el contingut del comptador.

Després d'haver fet el seguiment es pot veure que el que fa és comprovar per cada valor entrat si és múltiple de 2 i en cas de ser-ho augmenta en 1 el comptador, per tant al final mostrarà quants números parells ens han entrat per teclat.

Amb aquests exemples s'ha vist que es va repetint un fragment de codi fins que s'acaba l'entrada de valors, marcada per l'entrada d'un zero. Es proposa a continuació un exemple on serà possible que s'hagi de sortir del bucle abans d'haver avaluat tots els valors d'entrada:

*Es demana un programa que permeti comptar quants nucleòtids hi ha abans de trobar el nucleòtid T en una seqüència de nucleòtids acabada en *.*

Pensant la solució :

1. Inicialitzar el comptador.

2. Llegir el primer nucleòtid.

3. Comprovar que no sigui T ni tampoc *, perquè en cas de ser-ho ja cal anar a mostrar la quantitat de nucleòtids trobats abans de T.

4. Si es compleix la condició del punt 3:

- Cal augmentar el comptador.
- Cal llegir el següent nucleòtid.
- Tornar al punt 3.

5. Si ja no es compleix el punt 3 es mostra el resultat del comptador i s'acaba el programa.

Codificant la solució pensada:

```
compt=0
print("entra nucleotid")
nuc=input()
while nuc!="*" and nuc!="T" :
      compt=compt+1
      nuc=input()
print(compt)
```

ULL!!!, la condició posada al WHILE és composta, és a dir, té en compte dues coses, que no s'acabi la seqüència i que tampoc sigui T, doncs són les dues condicions que marquen fins quant s'han de comptar nucleòtids.

FOR..IN

Aquesta estructura permet que un fragment de codi s'executi un número determinat de vegades. És una estructura que no es pot usar si no se sap, prèviament a l'execució, el número d'iteracions que es realitzaran.

La seva sintaxi és:

```
for variable in range(valor_inicial, valor_final):
      instrucció1
      instrucció2
            :
            :
```

Si no es posa valor_inicial dins el range, la variable comença prenent el valor 0.

El seu funcionament és:

1. La variable pren valor_inicial, que es troba dins de RANGE
2. S'executen les instruccions de dins el FOR
3. Augmenta en 1 el valor de la variable
4. S'executen les instruccions de dins el for
5. Augmenta automàticament en 1 el valor de la variable

:
:

Es va repetint fins que la variable pren el valor_final definit en RANGE. Quan pren aquest valor ja no hi ha iteració.

Per conèixer el seu funcionament, es proposa el següent exemple:

Donada una seqüència de 100 nucleòtids dir quants n'hi ha de G.

Pensant la solució:

1.Inicialitzar el comptador

2.Llegir el nucleòtid

3.Si el nucleòtid és G cal que s'augmenti el comptador

4. Tornar a llegir un nucleòtid i passar al punt 3 si no s'han llegit ja 100 nucleòtids.

Si no s'hagués explicat el FOR, al veure que hi havia un fragment de codi que s'havia d'executar més d'un cop s'hauria fet amb un WHILE, posant com a condició d'iteració que no s'haguessin llegit ja els 100 nucleòtids.

Per fer això caldria disposar d'una variable que inicialment valdria 0 i que, a mesura que s'anés llegint un nucleòtid, augmentés en 1, i així quant ja fos més gran que 100 es sabria que ja s'ha processat tota la seqüència. El programa corresponent seria:

```
compt=0
i=0
print("entra nucleotid")
nuc=input()
i=i+1
while i<=100 :
      if nuc="g":
            compt=compt+1
      nuc=input()
      i=i+1
print(compt)
```

Escrit ara usant el FOR es veu molt més simple:

```
compt=0
for i in range(0, 100):
nuc=input()
      if nuc="G"
            compt=compt+1
print(compt)
```

1. S'inicialitza el comptador
2. Dins el FOR hi ha la variable i, que inicialment pren el valor 1 i que, per cada iteració augmentarà en 1 fins a l'última iteració que valdrà 100.
3. Les instruccions de dins el for llegeixen el nucleòtid i si és G augmenten en 1 el comptador
4. Sortint del FOR l'únic que queda per fer és mostrar el comptador on hi ha el valor corresponent al número de nucleòtids G que s'han trobat en la seqüència.

Un altre exemple:

Donat un número es demana que s'escrigui la seva taula de multiplicar.

Primer pensant la solució:

1. Cal demanar el número

2.Cal multiplicar el número entrat començant per l'1 i acabant per 10. Aquesta acció de multiplicació s'ha de repetir un número definit de vegades, per tant com a estructura iterativa s'escollirà el for. Cada resultat obtingut en la multiplicació es mostrarà.

```
print(" entra el número del que vols fer la seva taula de multiplicar")
num=input()
num=int(num)
for i in range(1, 11):
     res=num*i
print(res)
```

La variable que hi ha en el for, com que comença prenent el valor de 1 i va augmentant de un en 1 ja s'usa com a element a multiplicar per número i obtenir el resultat de la taula de multiplicar.

Després de llegir el número del que es vol mostrar la seva taula de multiplicar, s'executa la funció int(). En aquest cas s'ha usat la sentència input, per tant el tipus de dada entrat es tracta com una cadena, i interessa que sigui un valor enter. Per convertir-lo s'aplica l'esmentada funció.

Resum de la unitat didàctica

En aquesta unitat s'ha volgut donar una visió general de que és PYTHON i com s'escriu un programa en aquest llenguatge.

S'han explicat les sentències bàsiques per rebre dades entrades per teclat i per mostrar dades, s'ha introduït el concepte de variable, s'han definit els diferents tipus de dades elementals amb les que treballa, juntament amb els operadors associats i finalment s'han explicat quines són les sentències de control que marquen el flux de funcionament d'un programa.

Exercicis d'autoavaluació

1. Entrades per teclat l'acceleració i el temps d'un mòbil que segueix un MRUA es demana un programa que calculi l'espai recorregut i la velocitat adquirida, suposant que parteix del repòs.
2. Fer un programa que donats el pes molecular d'un compost i els grams que es tenen d'aquell compost, doni quin és el número de mols que tenim del compost.
3. Fer un programa que donat el símbol d'un nucleòtid en retorni el seu nom.
4. Fer un programa que donada una seqüència de nucleòtids acabada en *, comptar quants n'hi ha de cada tipus.
5. Fer un programa que donada una seqüència de nucleòtids acabada amb *, digui quants nucleòtids hi ha abans de trobar el doblet TA.
6. Fer un programa que donades 10 mesures preses en un laboratori digui quin és el valor més petit i el més gran. Remarcar que tota mesura és superior a 0 i inferior a 100.

Solucionari d'autoavaluació

1. Entrades per teclat l'acceleració i el temps d'un mòbil que segueix un MRUA es demana un programa que calculi l'espai recorregut i la velocitat adquirida, suposant que parteix del repòs.

```
print(" entra acceleracio:")
a=input()
a=float(a)
print("entra el temps")
t=input()
t=float(t)
s=1/2*a*t**2
v=a*t
print(" espai " + str(s))
print("velocitat " + str(v))
```

2. Fer un programa que donats el pes molecular d'un compost i els grams que es tenen d'aquell compost, doni quin és el número de mols que tenim del compost.

```
print(" entra pes molecular:")
p=input()
p=float(p)
print("entra els grams")
g=input()
g=float(g)
m=g/p
print(" el numero de mols es:" + str(m))
```

3.Fer un programa que donat el símbol d'un nucleòtid en retorni el seu nom.

```
print(" entra el nucleotid:")
nuc=input()
if  nuc="A" :
      print(" adenina")
elif nuc="G":
      print(" guanina")
elif nuc="C":
      print(" citosina")
elif nuc="T":
      print(" timina")
else:
     print("no es un nucleòtid")
```

4. Fer un programa que donada una seqüència de nucleòtids acabada en *, comptar quants n'hi ha de cada tipus.

```
compt=0
compa=0
compg=0
compc=0
print(" entra el nucleotid:")
nuc=input()
while nuc!="*":
      if  nuc="A" :
            compa=compa+1
      elif nuc="G":
            compg=compg+1
      elif nuc="C":
            compc=compc+1
      else
            compt=compt+1
      nuc=input()
print(" d'A:" + str(compa))
print(" d'G:" + str(compg))
print(" d'C:" + str(compc))
print(" d'T:" + str(compt))
```

5.Fer un programa que donada una seqüència de nucleòtids acabada amb *, digui quants nucleòtids hi ha abans de trobar el doblet TA.

```
compt=0
print(" entra el nucleotid:")
nuc=input()
if nuc!='*':
    ant=nuc
    print(" entra el nucleotid:")
    nuc=input()
    while nuc!="*" and (ant!="T" or nuc!="A"):
           compt=compt+1
           ant=nuc
           print("entra nucleotic:")
           nuc=input()
print("abans de TA n'hi havien: " + str(compt))
```

6.Fer un programa que donades 10 mesures preses en un laboratori digui quin és el valor més petit i el més gran. Remarcar que tota mesura és superior a 0 i inferior a 100 .

```
max= 0
min=100
for i in range(0, 10):
      print(" entra mesura")
      num=input()
      num=float(num)
      if num<min:
            min=num
      if num>max:
            max=num
print("la mesura mes petita:" + str(min))
print("la mesura mes gran:" + str(max))
```

Glossari de termes

Estructura de control: permet indicar al programa l'ordre en que s'executen les accions.
Estructura condicional: controla l'execució d'un fragment de codi depenen de si es compleix una condició.
Estructura iterativa: controla que es repeteixi l'execució d'un fragment de codi, sempre hi quan es vagi complint una condició.
Indentació: permet marcar el conjunt de instruccions que s'han d'executar dins d'una estructura de control.
Instrucció: sentència a realitzar dins un programa escrita en un llenguatge de programació**.**
Programa: un conjunt d'instruccions escrites en un llenguatge de programació que la CPU de l'ordinador va executant per tal de realitzar una tasca determinada.
PYTHON: llenguatge de programació**.**
Tipus de dades cadena: text delimitat per cometes simples ('hola') o dobles ("hola").
Tipus de dades numèric: enters, reals i complexos.
Variable: Espai de memòria destinat a guardar alguna dada.

Bibliografia recomanada

Lutz, M.;Ascher, D. Learning PYTHON. O'Reilly, 2a Edició
Model, M.; Bioinformatics Programming using PYTHON. O'Reilly.