Unutar treće cjeline nema programskih odsječaka sve do lekcije 3.3.16 (Pravila umnoška i zbroja se prirodno pojavljuju u programiranju). Pogledajmo sljedeća dva koda (u Python-u) u kojima se izvršava m for petlji s time da su u lijevo navedenom kodu one ugniježdene, dok su u desno navedenom kodu navedene jedna za drugom.

Prva petlja (ugnježdene)

```
k = 0
for i1 in range(1, n1 + 1):
    for i2 in range(1, n2 + 1):
        # ...
        for im in range(1, nm + 1):
            k += 1

##### Druga petlja (1 ispod druge)
k = 0
for i1 in range(1, n1 + 1):
    k += 1
for i2 in range(1, n2 + 1):
    k += 1
# ...
for im in range(1, nm + 1):
    k += 1
```

Vidimo da se svaka od for petlji izvršava ni puta, i {1, 2, . . . , m}, pri čemu se desne petlje izvršavaju neovisno jedna o drugoj. Stoga će vrijednost od k nakon izvršenja svih petlji biti jednaka n1 + n2 + . . . + nm, to je u skladu s pravilom zbroja. S druge strane, kod ugniježdenih petlji će se svaka unutrašnja petlja izvršiti onoliko puta koliko joj petlji prethodi pomnoženo s pripadnim ni, odnosno vrijednost od k nakon izvršenja svih petlji će biti jednaka n1 · n2 · . . . · nm, to je u skladu s pravilom umnoška.

Nadalje, programski se kodovi pojavljuju u lekciji 3.5. Kako su u kombinatorici veoma važni izračuni faktorijela i binomnih koeficijenata, logično je očekivati da u Pythonu postoji ugrađeni način da se izračunaju. Da bismo u Pythonu dobili mogućnost za računanje faktorijela i binomnog koeficijenta, možemo koristiti biblioteku math. Ovdje je Python kod za oba izračuna:

```
import math

# Računanje faktorijela broja 15
fact_15 = math.factorial(15)
print(fact_15)

# Izračun binomnog koeficijenta "10 nad 4"
binomial_coef = math.comb(10, 4)
print(binomial_coef)
```

Naravno, možemo izračunati razne vjerojatnosti i jesmo li izračunali ispravno možemo provjeriti pomoću generiranja slučajnih uzoraka. Ako želimo generirati uređeni uzorak od 5 brojeva od 10 dostupnih brojeva bez ponavljanja, koristimo sljedeći kod:

```
import random
```

```
# Generiraj uzorak od 5 brojeva od 1 do 10 bez ponavljanja
uzorak = random.sample(range(1, 11), 5)
print(uzorak)
```

Ovdje smo generirali uzorak 5 brojeva od 1 do 10. Sada možemo koristiti funkciju set.seed() za postavljanje početnog stanja generatora slučajnih brojeva kako bismo rezultate simulacije reproducirali na različite načine.

```
import random

# Postavi sjeme generatora slučajnih brojeva na 1
random.seed(1)

# Generiraj uzorak od 5 brojeva od 1 do 10 bez ponavljanja
uzorak1 = random.sample(range(1, 11), 5)
print(uzorak1)

# Postavi sjeme generatora slučajnih brojeva na 2022
random.seed(2022)

# Generiraj drugačiji uzorak od 5 brojeva od 1 do 10 bez ponavljanja
uzorak2 = random.sample(range(1, 11), 5)
print(uzorak2)
```

Ukoliko želimo kreirati veći broj slučajnih vrijednosti, možemo unutar naše sample funkcije uključiti for-petlju:

```
import random

# Generiranje 13 uzoraka od po 5 nasumičnih brojeva od 1 do 10 bez ponavljanja
uzorci = [random.sample(range(1, 11), 5) for _ in range(13)]
```

```
import random

# Generiranje 13 uzoraka od po 5 nasumičnih brojeva od 1 do 10 s ponavljanjem
uzorci_s_ponavljanjem = [random.choices(range(1, 11), k=5) for _ in range(13)]
```

Na sličan način možemo iz niza koji sadrži slova, generirati slučajan uzorak proizvoljne duljine. Primjera radi, odabrat ćemo duljinu 5.

```
import random

# Generiranje uzorka znakova duljine 5 iz niza znakova a, b, c, d, e, f, g, h, i, j
niz_znakova = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j"]
uzorak_znakova = random.choices(niz_znakova, k=5)

# Ispis uzorka
print(uzorak_znakova)
```

Uz malo razmišljanja u Pythonu možemo izračunati i vjerojatnost da barem dvoje od n ljudi ima rođendan na isti dan. Uzet ćemo 23 osobe za naš primjer:

```
# Broj ljudi
k = 23

# Izračun vjerojatnosti da barem dvoje od 23 ljudi ima isti rođendan
p = 1 - (365 / 365)  # Početna vjerojatnost da nema zajedničkog rođendana

for i in range(k):
    p *= (365 - i) / 365  # Izračun vjerojatnosti da nema zajedničkog rođendana za svaku osobu

p = 1 - p  # Konačna vjerojatnost da barem dvoje ljudi imaju isti rođendan

# Ispis rezultata
print("Tražena vjerojatnost je jednaka", p)
```

U R-u postoje određene funkcije za izračun vjerojatnosti rođendana. Primjerice, kolike su šanse da od skupa odabranih osoba, bar 2 imaju rođendan na isti dan. U Pythonu one nisu ugrađene, ali svejedno ih možemo korisnički definirati. (NAPOMENA: print f je Python funkcija koja korisniku omogućuje ispis formatiranog stringa koji će se mijenjati ovisno o vrijednosti pojedine varijable koju ispisujemo)

```
# Funkcija za izračun vjerojatnosti da će barem dvije osobe (od ukupno jako puno) imati rođendan istog dana.
def vjerojatnost_rođendana(k):
    p = 1
    for i in range(k):
        p *= (365 - i) / 365
    return 1 - p

# Funkcija za izračun minimalnog broja ljudi potrebnog za postizanje vjerojatnosti od 50% da barem dvije osobe imaju rođendan istog dana.
def minimalni_broj_osoba(vjerojatnost):
    k = 0
    p = 1
    while p > 1 - vjerojatnost:
        k += 1
        p *= (365 - k) / 365
    return k

# Izračun vjerojatnosti za 23 osobe
vjerojatnost_za_23_osobe = vjerojatnost_rođendana(23)
print(f"Vjerojatnost da će barem dvije osobe od 23 imati rođendan istog dana je: {vjerojatnost_za_23_osobe}")

# Izračun minimalnog broja osoba za vjerojatnost od 50%
minimalan_broj_osoba = minimalni_broj_osoba(0.5)
print(f"Minimalan broj osoba potreban za vjerojatnost od 50% da barem dvije osobe imaju rođendan istog dana je: {minimalan_broj_osoba}")
```

Također, moguće je vizualno prikazati dobivene podatke pomoću grafova. Python koristi biblioteku Matplotlib za izradu grafa i matematičke operacije za izračun

vjerojatnosti. Rezultat je graf koji prikazuje promjenu vjerojatnosti za različite brojeve ljudi.

```
import matplotlib.pyplot as plt
import math

# Inicijalizacija vektora za pohranu vjerojatnosti
p = []

# Izračun vjerojatnosti za svaki broj ljudi od 1 do 100
for k in range(1, 101):
    vjerojatnost = 1 - math.prod([(365 - i) / 365 for i in range(k)])
    p.append(vjerojatnost)

# Kreiranje grafa
plt.figure(figsize=(8, 6))
plt.plot(range(1, 101), p, marker='o', markersize=4, linestyle='-')
plt.xlabel("Broj ljudi")
plt.ylabel("Vjerojatnost višestrukih rođendana")
plt.title("Vjerojatnost višestrukih rođendana u ovisnosti o broju ljudi")
plt.grid(True)
plt.show()
```

Nadalje, možemo i generirati slučajne rođendane za određeni broj ljudi. Konkretno, kreirat ćemo slučajne rođendane za 23 osobe. Kako se radi o rođendanima, nije zabranjeno da se vrijednosti ponavljaju.

```
import random

# Postavljanje početnog stanja generatora slučajnih brojeva (za reproduktivnost)
random.seed(42)

# Generiranje uzorka rođendana za 23 ljudi s ponavljanjem (moguće su iste vrijednosti)
rodjendani = [random.randint(1, 365) for _ in range(23)]

# Ispis uzorka rođendana
print(rodjendani)
```

Za kraj, možemo demonstrirati i rad s velikim brojem simulacija. Naime, radi postizanja boljeg uzorka, možemo izvesti 100000 simulacija i izračunati vjerojatnost da od 100000 ljudi, bar dvije dijele isti rođendan.

```
import random
import numpy as np

# Postavljanje početnog stanja generatora slučajnih brojeva (za reproduktivnost)
random.seed(42)

# Generiranje velikog broja simulacija
broj_simulacija = 10**5
simulacije = [max(np.bincount([random.randint(1, 365) for _ in range(23)])) for _ in
range(broj_simulacija)]
```

```
# Izračun postotka simulacija u kojima barem dvije osobe dijele isti rođendan
postotak_simulacija_s_dvostrukim_rodjendanom = sum(np.array(simulacije) >= 2) /
broj_simulacija

# Ispis rezultata simulacije
print(postotak_simulacija_s_dvostrukim_rodjendanom)

# Ispis vjerojatnosti izračunate pomoću pbirthday
vjerojatnost_rođendana = 1 - np.prod(1 - np.arange(1, 366) / 365)
print(vjerojatnost_rođendana)
```