

Za radoznale: Funkcije i moduli

Ovaj fajl je za one koji žele više. Ništa odavde nije potrebno za praćenje časova. Svaki deo počinje primerom koji smo radili na času, a zatim pokazuje kako se ista stvar može zapisati kraće ili naprednije. Svi primeri su i u folderu primeri/za_radoznale, pa ih možete pokrenuti, menjati i gledati šta se dešava.

1. *args: proizvoljan broj argumenata

Na času je svaka funkcija imala tačno određen broj parametara. Funkcija pozdrav sa dva parametra mora da dobije tačno dva argumenta:

```
def pozdrav(ime, prezime):  
    print(f"Ja sam {ime} {prezime}!")  
  
pozdrav("Pera", "Perić")
```

Ispis:

```
Ja sam Pera Perić!
```

Ponekad ne znamo unapred koliko će argumenata funkcija dobiti, na primer koliko drugova pozdravljamo. Tada ispred imena parametra stavimo zvezdicu, pa se svi argumenti skupe u jednu torku. U funkciji suma prvi parametar je običan i obavezan, a svi ostali argumenti odlaze u torku ostale_vrednosti. Oba primera su iz starih beleški.

```
def pozdrav(*drugovi):  
    print("Drugovi:", drugovi)  
  
pozdrav("Pera", "Mika")  
pozdrav("Pera", "Mika", "Laza")  
  
def suma(pocetna_vrednost, *ostale_vrednosti):  
    zbir = pocetna_vrednost  
    for broj in ostale_vrednosti:  
        zbir += broj  
    return zbir  
  
print(suma(0))  
print(suma(0, 1, 2))  
print(suma(10, 5, 5, 5))
```

Ispis:

```
Drugovi: ('Pera', 'Mika')  
Drugovi: ('Pera', 'Mika', 'Laza')  
0  
3  
25
```

Poziv suma(0) daje 0, jer je torka ostale_vrednosti prazna, pa se petlja ne izvrši nijednom. Ugrađena funkcija print radi baš ovako: zato joj možemo dati koliko god hoćemo argumenata.

Zašto ovo nismo radili na času: Parametar sa zvezdicom traži da dobro razumemo i torke i obične parametre. Na prvom času o funkcijama bilo je važnije da utvrdimo parametre, podrazumevane vrednosti i imenovane argumente, koji nam trebaju u PyGame-u.

2. ****kwargs: proizvoljan broj imenovanih argumenata**

Na času smo videli imenovane argumente i podrazumevanu vrednost:

```
def pozdrav(ime, prezime, broj_godina=30):  
    print(f"Ja sam {ime} {prezime}. Imam {broj_godina} godina.")  
  
pozdrav(prezime="Perić", broj_godina=20, ime="Pera")
```

Ispis:

```
Ja sam Pera Perić. Imam 20 godina.
```

Sa dve zvezdice funkcija prima koliko god hoćemo imenovanih argumenata, čak i sa imenima koja nismo unapred napisali. Svi oni se skupe u rečnik: ime argumenta je ključ, a vrednost argumenta je vrednost. Zgodno je za lika u igri, jer svaki lik može da ima druge osobine.

```
def lik(ime, **osobine):  
    print("Lik:", ime)  
    for kljuc, vrednost in osobine.items():  
        print(" ", kljuc, "=", vrednost)  
  
lik("Cikica", zivot=3, brzina=2)  
lik("Bubica", oruzje="luk")
```

Ispis:

```
Lik: Cikica  
    zivot = 3  
    brzina = 2  
Lik: Bubica  
    oruzje = luk
```

Zašto ovo nismo radili na času: Ovde se spajaju funkcije i rečnici, a za takav spoj nam je trebalo vremena da se oba pojma slegnu. Za sve naše primere dovoljni su obični parametri.

3. Zvezdice u pozivu: raspakivanje torke i rečnika

Na času smo funkciji zvezda davali argumente jedan po jedan:

```
def zvezda(x, y, velicina=3, boja="bela"):  
    print("zvezda", x, y, velicina, boja)  
  
zvezda(10, 20, boja="žuta")
```

Ispis:

```
zvezda 10 20 3 žuta
```

Kad su položaj i izgled već sačuvani u torki i rečniku, ne moramo da ih vadimo deo po deo. Zvezdica u pozivu raspakuje torku u argumente po redu, a dve zvezdice raspakuju rečnik u imenovane argumente.

```
def zvezda(x, y, velicina=3, boja="bela"):
    print("zvezda", x, y, velicina, boja)

polozaj = (10, 20)
izgled = {"velicina": 5, "boja": "žuta"}

zvezda(*polozaj)
zvezda(*polozaj, **izgled)
```

Ispis:

```
zvezda 10 20 3 bela
zvezda 10 20 5 žuta
```

Zašto ovo nismo radili na času: Isti znak, zvezdica, ovde radi suprotno od onoga što radi u definiciji, pa se lako pomeša. Ima smisla tek kad su *args* i **kwargs* već jasni.

4. lambda: funkcija bez imena kao argument

Na času smo funkciji racunarska_operacija prosledili funkciju koju smo prethodno napisali sa def:

```
def racunarska_operacija(operacija, broj):
    return operacija(broj)

def uvecanje_za_1(broj):
    return broj + 1

print(racunarska_operacija(uvecanje_za_1, 5))
```

Ispis:

```
6
```

Kad je funkcija kratka i treba nam samo na jednom mestu, možemo da je napišemo u samom pozivu, rečju lambda. Zapis lambda broj: broj * 10 znači „funkcija koja za broj vraća broj * 10“. Isto radi i kao argument key kod sortiranja: ovde reći sortiramo po poslednjem slovu.

```
def racunarska_operacija(operacija, broj):
    return operacija(broj)

print(racunarska_operacija(lambda broj: broj * 10, 5))

reci = ["platno", "kist", "boja", "četskica"]
print(sorted(reci, key=lambda r: r[-1]))
```

Ispis:

```
50
['boja', 'četskica', 'platno', 'kist']
```

Zašto ovo nismo radili na času: Funkcija napisana sa def ima ime i lakše se čita, a lambda je samo kraći zapis iste ideje. Prvo smo hteli da dobro razumemo da je funkcija vrednost koja se može proslediti.

5. Opis funkcije: docstring

Na času smo napisali funkciju površina:

```
def površina(sirina, visina):  
    return sirina * visina  
  
print(površina(1920, 1080))
```

Ispis:

```
2073600
```

Ako odmah ispod reda sa def napišemo string u trostrukim navodnicima, on postaje opis funkcije (engl. docstring). Python ga pamti uz funkciju, a razvojna okruženja ga prikazuju kad pređemo mišem preko imena funkcije. Opis možemo i sami da ispišemo.

```
def površina(sirina, visina):  
    """Vraća površinu pravougaonika date širine i visine."""  
    return sirina * visina  
  
print(površina(1920, 1080))  
print(površina.__doc__)
```

Ispis:

```
2073600  
Vraća površinu pravougaonika date širine i visine.
```

Zašto ovo nismo radili na času: Za kratke funkcije na času dobro izabrano ime govori dovoljno. Opis postaje važan kad pišemo module koje koriste i drugi ljudi.

6. Modul koji može i da se pokrene: name

Na času smo napravili modul paleta.py i uvezli ga u drugi program. Svaki program ima ugrađeno ime **name**, koje kaže kako je fajl pokrenut:

```
print(__name__)
```

Ispis:

```
__main__
```

Kad fajl pokrenemo direktno, **name** ima vrednost **"main"**. Kad isti fajl uvezemo naredbom import, **name** je ime modula, na primer "paleta". Zato u modul možemo da dopišemo mali probni deo, koji se izvršava samo kad modul pokrenemo, a ne smeta kad ga uvezemo.

```
# paleta.py, sa probnim delom na kraju
CRNA = (0, 0, 0)

def posvetli(boja, koliko):
    r, g, b = boja
    return (min(r + koliko, 255), min(g + koliko, 255), min(b + koliko, 255))

if __name__ == "__main__":
    print("Probam modul:", posvetli(CRNA, 60))
```

Ispis:

```
Probam modul: (60, 60, 60)
```

Kad drugi program napiše `import paleta`, probni red se ne izvršava i ništa se ne ispisuje.

Zašto ovo nismo radili na času: Naši moduli su bili mali i nismo ih pokretali sami. Ova provera postaje korisna tek kod većih modula, koje želimo da isprobamo i pre nego što ih uvezemo.