3-2-Funkcije-S

Mogućnost definiranja novih funkcija je osnovna stepenica k naprednijem programiranju. U Sageu postoji nekoliko vrsta funkcija (za detalje vidi ovdje), a mi ćemo trebati i razlikovati dvije osnovne vrste:

1. Simboličke funkcije
2. Python funkcije

Ugrubo, simbolička funkcija dopušta više simboličkih manipulacija (poput integracije ili deriviranja), ali ne može sadržavati kompleksne algoritme već samo jednostavne izraze. Python funkcija može biti proizvoljno kompleksna, ali ne može se uvijek npr. simbolički integrirati.

Simboličke funkcije se definiraju na slijedeći prirodan način:

```
f(x) = x^2
f
```

```
x |--> x^2
```

S njima se mogu raditi sve uobičajene operacije:

```
print f(2)
print f((2*x+3)^2)
diff(f(x), x)
```

```
4
(2*x + 3)^4
2*x
```

(*) Formalno, simboličke funkcije nisu po svom tipu "funkcije", već "simbolički izrazi koji se mogu pozivati" ("*callable symbolic expression*"):

```
print type(sin)
type(f)
```

```
<class 'sage.functions.trig.Function_sin'>
<type 'sage.symbolic.expression.Expression'>
```

Naravno, funkcija može biti i funkcija od više variabli:

Naravno, funkcija može biti i funkcija od više variabli:

```
g(x, a) = x^a
g(4, 2)
```

16

Python funkcije se definiraju korištenjem ključnih riječi *def* i *return*, te blokova kôda koji su konzistentno uvučeni (npr. za 4 mjesta; Sage automatski radi to uvlačenje):

```
def h(x):
    "Kvadriraj broj x."
    return x^2
```

```
print h(3)
```

9

Kao prvi red tijela funkcije može se, kao gore, staviti dokumentacijski string (tzv. *docstring*) kojem se kasnije može pristupiti standardnim metodama pristupa dokumentaciji:

```
h?
```

File: /var/tmp/tmp_M4ITk/___code___.py

Type: <type 'function'>

Definition: h(x)

Docstring:

Kvadriraj broj x.

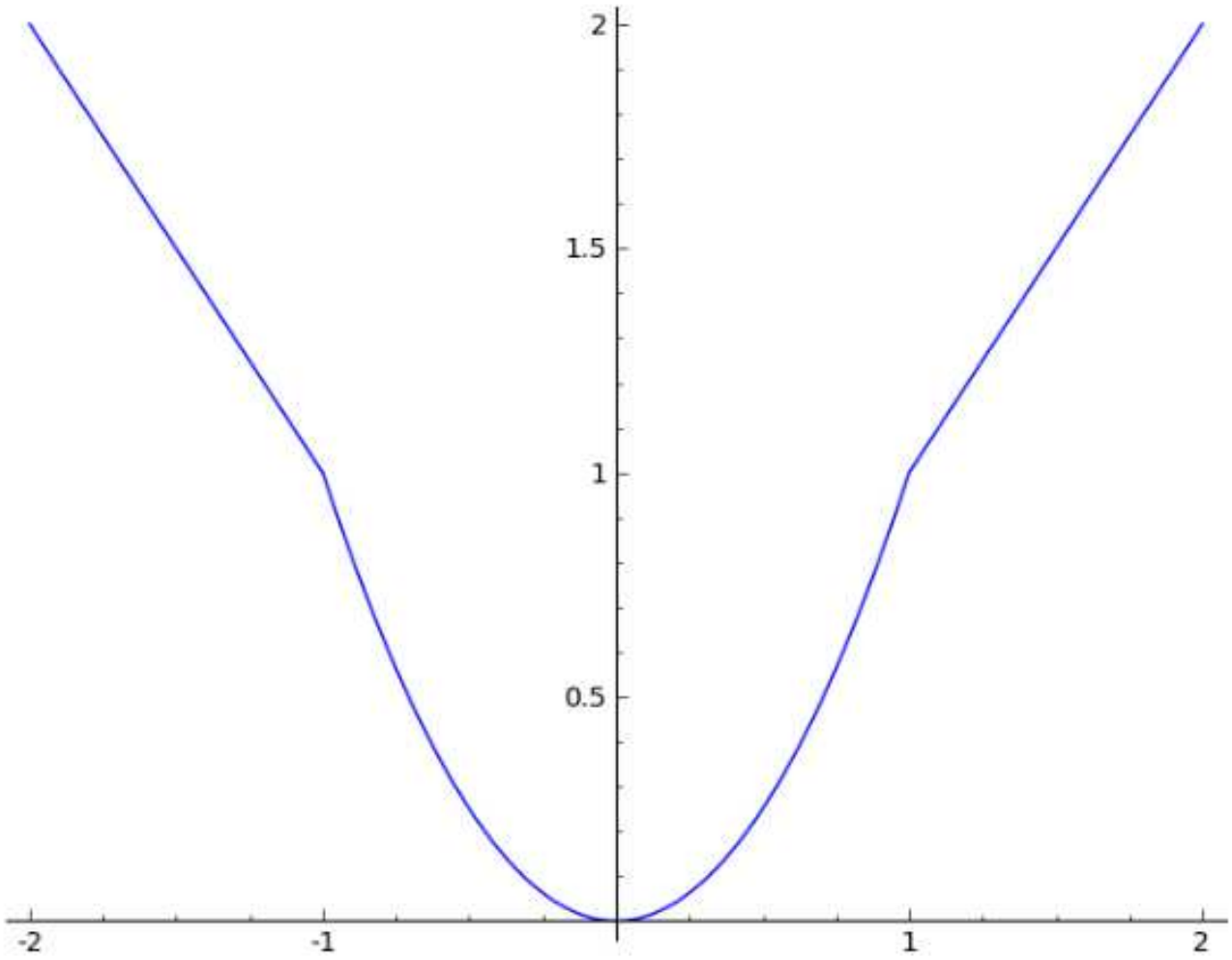
Funkciju definiranu po dijelovima možemo dobiti na slijedeći način:

```
def fpw(x):
    if x<-1:
        return -x
    elif x>1:
```

```
        return x
    else:
        return x^2
```

(Ovdje smo upotrijebili if-then-else grananje. Sintaksa je očita.)

```
plot(fpw, -2, 2)
```



Kako je već spomenuto, python funkcije ne možemo općenito npr. simbolički integrirati. Posebno je opasno to što, ukoliko pokušamo, možemo dobiti pogrešan odgovor:

```
integral(fpw(x), x, 1, 2)
```

```
7/3
```

No, uvijek možemo pribjeći numeričkoj integraciji koja u ovom slučaju daje točan rezultat 3/2:

```
numerical_integral(fpw, 1, 2)[0]
```

```
1.5
```

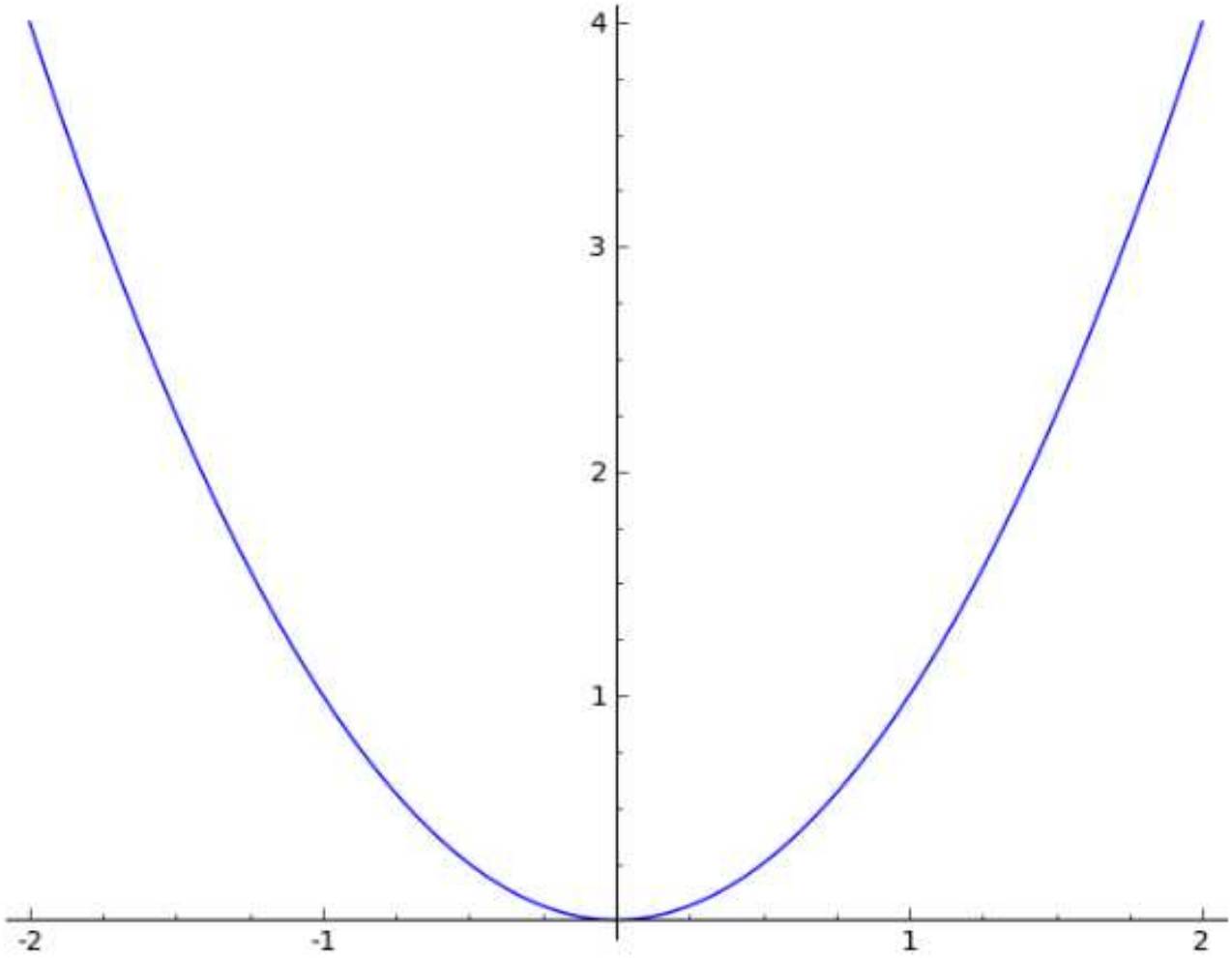
(*) Pri gornjem pokušaju simboličke integracije, prvo je izvrijednjen sam integrand fpw(x). Kako u tom trenutku x nema nikakvu vrijednost, on ne zadovoljava ni jedan od dva uvjeta (x>1 ili x<1) pa funkcija vraća simbolički izraz x^2 ("else" blok), koji je onda integriran u granicama od 1 do 2.

```
fpw(x)
```

```
x^2
```

Iz istog razloga je prilikom crtanja gore bilo potrebno kao prvi argument staviti samu funkciju fpw, a ne izraz fpw(x) koji bi dao krivi crtež:

```
plot(fpw(x), x, -2, 2)
```



Funkcija može imati i opcionalne argumente s defaultnom vrijednošću:

```
def fun(x, n=1, b=0):
    return x^n + b
```

```
print fun(3, 4, 5)
print fun(3, b=5, n=4)
fun(3)
```

```
86
86
3
```

(Uočite da kad smo eksplicitno imenovali argumente nismo morali paziti na njihov poredak.)

Bilo što može biti argument funkcije. Najmoćnija stvar, obilato korištena u funkcionalnom pristupu programiranju (vidi kasnije), je da i same funkcije mogu biti argumenti funkcija:

```
def gun(f, x):
    "Komponiraj dvaput funkciju sa samom sobom"
    return f(f(x))
```

```
print gun(sin, 2)
gun(log, 0.1)
```

```
sin(sin(2))
0.834032445247956 + 3.14159265358979*I
```

Ukoliko znamo da python funkcija uvijek korektno vraća simbolički izraz, smijemo je integrirati i diferencirati:

```
diff(gun(sin, x), x)
```

```
cos(x)*cos(sin(x))
```

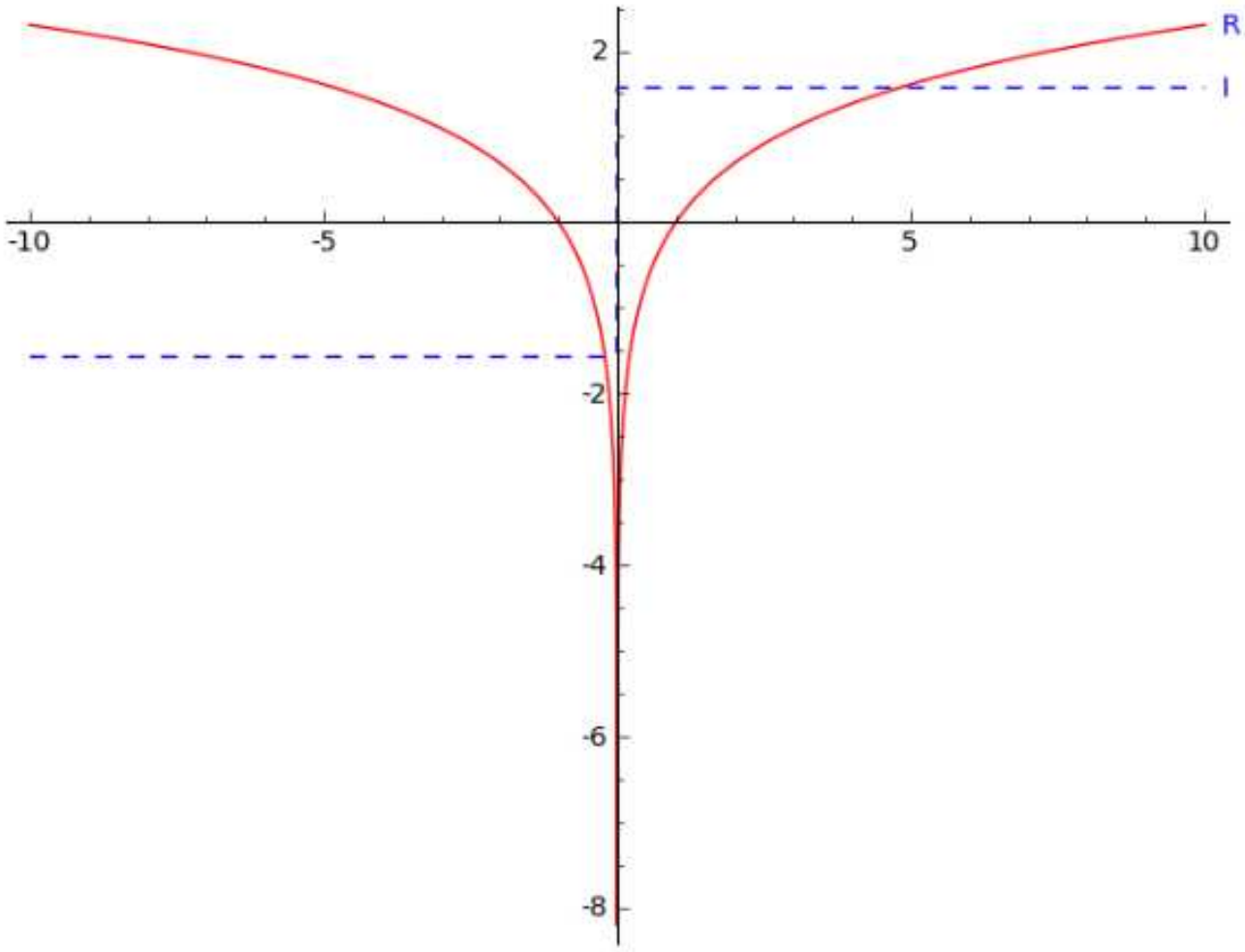
Rezultat izvrijednjavanja funkcije, dakle ono što slijedi ključnu riječ *return*, može biti bilo kakav objekt. Čak i crtež:

```
def plotfun(fun, min, max):
    "Nacrtaj realni i imaginarni dio funkcije fun."

    Pre = plot(real(fun(x)), min, max, color='red', label="R")
    Pim = plot(imag(fun(x)), min, max, color='blue',
linestyle='--', label="I")
    return Pre+Pim
```

```
def imlog(x):
    return log(I*x)
```

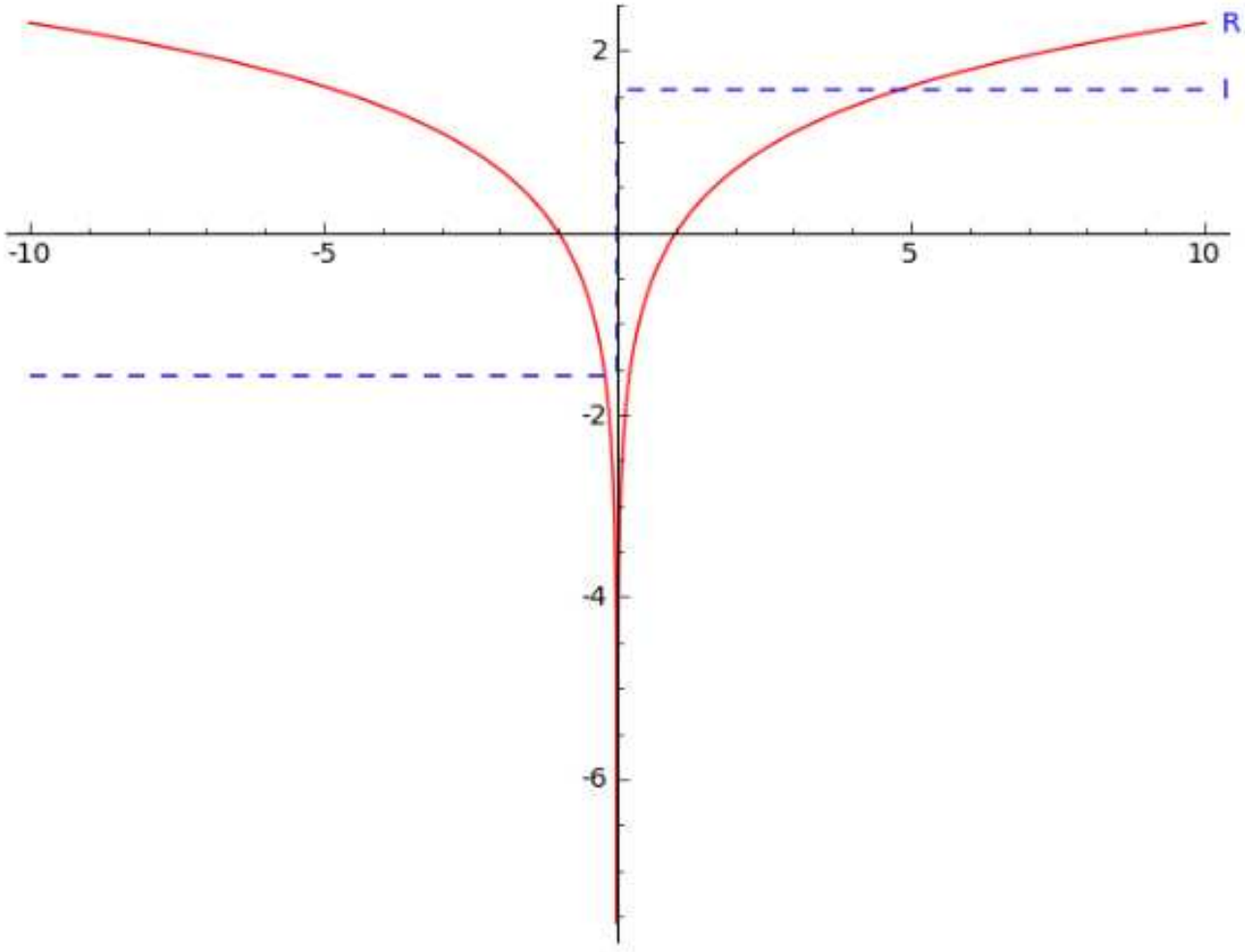
```
plotfun(imlog, -10, 10)
```



Može i ovo, ali nije poželjno:

```
plotfun(imlog(x), -10, 10)
```

```
__main__:1: DeprecationWarning: Substitution using function-call
syntax and unnamed arguments is deprecated and will be removed
from
a future release of Sage; you can use named arguments instead,
like
EXPR(x=..., y=...)
```



 Zadatak 3.2.1: Definirajte funkciju plotpoint(z) koja crta kompleksnu ravninu sa točkom koja odgovara kompleksnom broju z. Dakle x=Re(z), y=Im(z)

 Zadatak 3.2.2: Definirajte funkciju argand(lista) koja crta kompleksnu ravninu s točkama zadanim u listi kompleksnih brojeva lista = $[z_1, z_2, \ldots]$. Upotrijebite funkciju argand da nacrtate na jednom dijagramu svih devet devetih korijena od 1.

Zadatak 3.2.3: Formule za nerelativističku i relativističku kinetičku energiju tijela mase m koje se giba brzinom v su

$$K_{\mathrm{nr}} = \frac{1}{2}mv^2 \ , \qquad K_{\mathrm{rel}} = mc^2 \left(\frac{1}{\sqrt{1 - v^2/c^2}} - 1 \right) \ .$$

(a) Definirajte odgovarajuće funkcije *knr()* i *krel()* te usporedite na istom dijagramu ponašanje tih funkcija za brzine od 0 do $3 \cdot 10^8$ m/s za neku vrijednost mase m=1 kg. Označite veličine i njihove jedinice na koordinatnim osima!

(b) Odredite brzinu pri kojoj je greška nerelativističke formule tisućinku promila.

(c) Ako ste definirali funkcije kao python funkcije, ponovite zadatak sa simboličkim funkcijama i obratno.