Python - nie takie OOP straszne, jak je malują

> “Jeśli spędzisz dodatkową godzinę w każdym dniu,
> na doskonaleniu kierunku, który wybrałeś, będziesz ekspertem krajowym
> w tej dziedzinie w ciągu pięciu lat lub wcześniej.”
> ~Earl Nightingale

Dlaczego ryś jest jak chodnik? Ponieważ i ryś, i chodnik są “cośkami”. W języku polskim *coś* jest rzeczownikiem, w Pythonie jest *obiektem.*
Idea *obiektów* jest ważna w świecie komputerów. Obiekty to sposób na organizację kodu w programie i rozbicie rzeczy by ułatwić myślenie o złożonych problemach.

Żeby zrozumieć, jak *obiekty* działają w Pythonie, musimy pomyśleć o typach obiektów. Wróćmy do przykładu z rysiem i chodnikiem. Ryś jest typem ssaka, który jest typem zwierzęcia. Ryś jest też ożywionym obiektem - jest żywy.

Rozważmy teraz chodnik. Nie można o nim powiedzieć więcej, niż to, że nie jest żywy. Nazwijmy go więc nieożywionym obiektem.

Terminy *ssak*, *zwierzę, ożywiony* i *nieożywiony* to sposoby na sklasyfikowanie cośków.

Rozbijanie cośków na klasy

W Pythonie obiekty definiujemy jako *klasy,* które są sposobem na pogrupowanie naszych obiektów. Poniżej prezentujemy diagram drzewa klas, do których wpasują się nasz ryś i chodnik.

Główną klasą są Cośki. Pod klasą Cośki, mamy klasy Ozywione i Nieozywione. Te są rozbite na kolejne pomniejsze (jak widać na obrazku).

Możemy użyć klas do organizowania fragmentów kodu w Pythonie. Weźmy jako przykład moduł z poprzedniej lekcji - *turtle*. Wszystkie rzeczy, który moduł *turtle* potrafił zrobić - takie jak ruch do przodu, obrót w lewo, obrót w prawo - są funkcjami w klasie Pen.
Obiekt może być uważany za członka klasy (tak jak ryś jest członkiem klasy Rysie) i możemy stworzyć dowolną liczbę obiektów dla klasy (ryś1, ryś2, itd.), którą wkrótce otrzymamy.
Stwórzmy teraz ten sam układ klas jak na diagramie, zaczynając od początku. Definiujemy klasę używając słowa *class*, po którym piszemy jej nazwę:

```
class Coski:
    pass
```

Nazwaliśmy klasę Coski (czyt. cośki, nie używamy polskich znaków przy nazewnictwie zmiennych, klas, funkcji itp.) i użyliśmy wyrażenia *pass*, aby Python wiedział, że nie zamierzamy podawać więcej informacji.

pass jest używany, gdy chcemy zapewnić klasę lub funkcję, ale nie chcemy w tym momencie wypełniać danych.

Dalej dodamy kolejne klasy i stworzymy związki między nimi.

Rodzice i ich potomstwo

Jeśli klasa jest częścią innej klasy to jest dzieckiem tej klasy, a druga klasa jest jej rodzicem.
Klasy mogą zarówno być dziećmi jak i rodzicami innych klas. Na naszym diagramie wyżej klasa powyżej innej klasy jest rodzicem. Ożywione i nieożywione są dziećmi klasy rzeczy. Aby "powiedzieć" pythonowi, że klasa jest dzieckiem innej klasy dodajemy nazwę klasy nadrzędnej w nawiasach po nazwie naszej klasy:

```
class Ozywione(Cosiek):
    pass

class Nieozywione(Cosiek):
    pass
```

Tutaj tworzymy klasę o nazwie **Nieozywione** i mówimy Pythonowi, że jej klasa nadrzędna to **Cosiek**.

Tworzenie obiektu klasy

Mamy teraz kilka klas, ale co z umieszczaniem niektórych rzeczy do tych klas. Powiedzmy, że nasz ryś ma na imię Rysio. Wiemy, że on należy klasy **Rys**, ale czego użyjemy w pythonie, aby tego jednego rysia nazwać Rysio. W pythonie Rysio będzie nazywany obiektem klasy **Rys**. Zrobimy to tak:

```
Rysio = Rys()
```

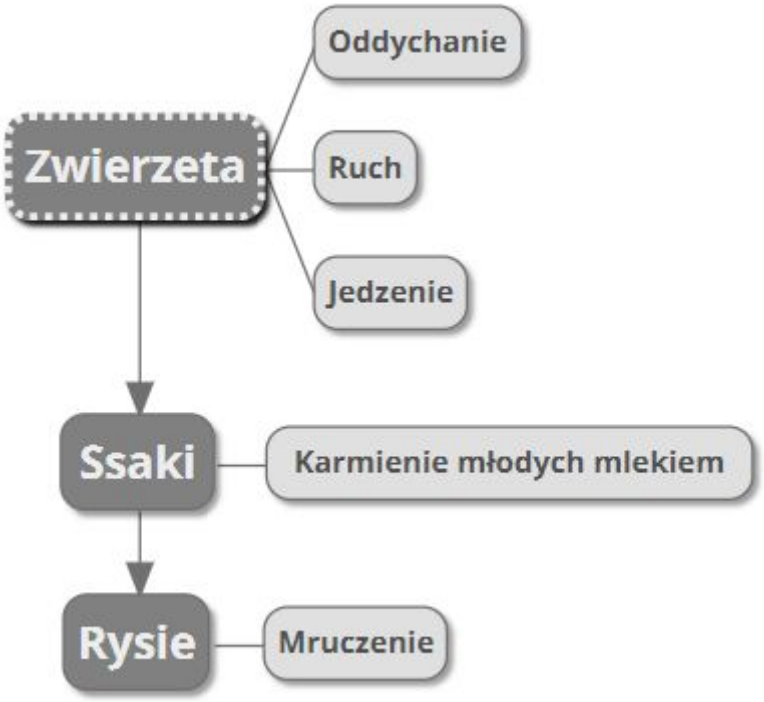
Ten kod mówi Pythonowi, aby utworzył obiekt klasy Rys i przypisał go do zmiennej Rysio. Tak samo jak przy funkcjach po nazwie klasy występuje nawias. W dalszej części notatki dowiemy się, jak tworzyć obiekty i używać parametrów w nawiasach. Ale co obiekt Rysio teraz robi? Cóż… na obecną chwilę nic. Aby nasze obiekty były przydatne, w trakcie tworzenia klasy musimy zdefiniować ich zawartość. Zamiast od razu używać słowa kluczowego pass po definicji klasy możemy dodać np. definicje funkcji.

Definiowanie funkcji w klasie

W poprzedniej notatce (Python - Początki 2) wprowadziliśmy funkcje jako sposób ponownego użycia kodu. W taki sam sposób definiujemy funkcje w klasie, z wyjątkiem tego, że musimy napisać ją poniżej definicji klasy. Przykład:

```
class Rys(Zwierzeta):
    def DajGlos(self):
        print('Daje glos')
```

Rozważmy dzieci klasy **Ozywione**. Możemy dodać cechy charakterystyczne dla każdej z klasy, aby opisać co to jest i co może zrobić. Cechą charakterystyczną jest coś wspólnego dla wszystkich członków klasy(obiektów) i jej dzieci. Na przykład, co wspólnego mają wszystkie zwierzęta? Na pewno wszystkie oddychają, poruszają się i jedzą. Dobra, to co z ssakami? Co je łączy? Wszystkie ssaki karmią swoje młode mlekiem. Na diagramie (na nast.str.) wygląda to tak:



Te cechy można uznać za działania lub funkcje “coskow”, która obiekty tej klasy mogą robić.
A więc dodajmy funkcję do klasy:

```
class Animals(Ozywione):
    def Oddychanie(self):
        pass
    def Ruch(self):
        pass
    def Jedzenie(self):
        pass
```

W pierwszym wierszu tego kodu definiujemy klasę tak jak wcześniej, ale zamiast używać słowa **pass** w następnym wierszu definiujemy funkcję Oddychanie i dajemy jej jeden parametr **self**. Parametr self jest sposobem, aby w klasie wywołać funkcję w klasie. Wrócimy do niego później. W następnej linii słowo kluczowe pass mówi Pythonowi, że nie będziemy dostarczać więcej informacji na temat oddychania - funkcja nic nie robi. Następnie dodajemy funkcję Ruch. Ta funkcja również nic nie robi. Jest to powszechny sposób na rozwijanie programów.

Często programiści tworzą klasy z funkcjami, które nic nie robią, aby dowiedzieć się, co klasy powinny zrobić, zanim przejdą do szczegółów każdej funkcji. Każda klasa może korzystać z funkcji swojego rodzica. Nazywamy to dziedziczeniem.

Dlaczego używamy klas i obiektów?

Właśnie dodaliśmy funkcje do naszych klas, ale tak właściwie to po co musimy korzystać z tych klas? Kiedy po prostu można napisać normalne funkcje o nazwie oddychać itd.

Aby odpowiedzieć na to pytanie użyjemy naszego rysia o nazwie Rysio, którego stworzyliśmy wcześniej jako obiekt klasy Rys w taki sposób:

```
Rysio = Rys()
```

Ponieważ Rysio jest obiektem, możemy wywołać (uruchomić) dostępne funkcje dostarczone przez klasę (klasa Rys) i jej klasy nadrzędne (rodziców). Funkcję wywołujemy za pomocą operatora kropki i nazwy funkcji. Możemy kazać Rysiowi jeść albo oddychać w taki sposób:

```
Rysio = Rys()
Rysio.Ruch()
Rysio.Oddychanie()
```

Załóżmy, że Rysio ma przyjaciela o imieniu Eustachy. Stwórzmy inny obiekt Rys i nazwijmy go Eustachy:

```
Eustachy = Rys()
```

Ponieważ używamy obiektów i klasy możemy powiedzieć pythonowi dokładnie, o którym rysiu mówimy. Na przykład jak chcemy zrobić ruch Eustachym to wywołujemy funkcję należącą do obiektu o takiej nazwie.
To teraz edytujemy nasze funkcje, żeby coś robiły.

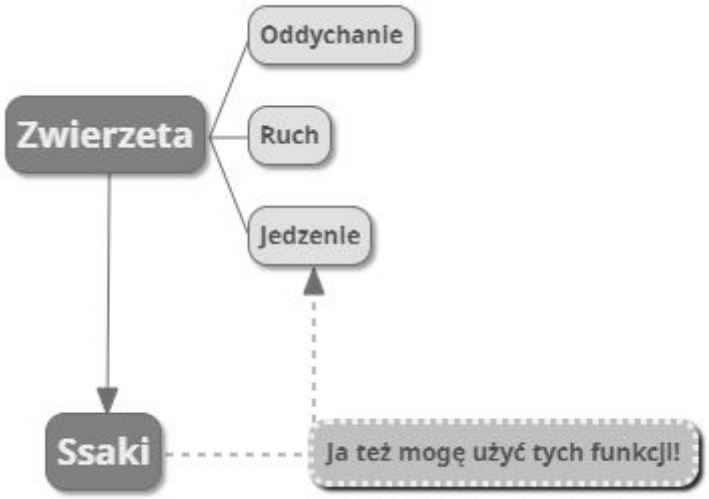
```
class Animals(Ozywione):
    def Oddychanie(self):
        print("Oddychanie")
    def Ruch(self):
        print("Ruch")
    def Jedzenie(self):
        print("Mlask, mlask")
```

Klasy i obiekty ułatwiają grupowanie funkcji oraz podział programu na mniejsze części. Weźmy jako przykład naprawdę dużą aplikację np. gra 3D. To prawie niemożliwe (dla większości ludzi), aby zrozumieć wszystko od razu. Możemy sobie poradzić z tym problemem dzieląc kod na mniejsze kawałki, a każdy kawałek staje się coraz łatwiejszy do zrozumienia - oczywiście jeśli znasz język.

Podczas pisania dużego programu, rozbicie go pozwala podzielić pracę na innych programistów. Wielu? Ale jak to? Sam nie stworzę GTA?! Jesteś w stanie to zrobić, ale zajmie Ci to znacznie więcej czasu. Najbardziej skomplikowane programy zostały napisane przez wielu ludzi lub zespoły ludzi, pracujących w różnych częściach świata.

Dziedziczenie

Ci z was, którzy zwracacie uwagę mogliście zauważyć, że każdy kto kończy pracę nad klasą Rys ma szczęście, ponieważ wszelkie funkcje stworzone przez ludzi pracujących nad klasami Zwierzęta i Ssaki mogą być wykorzystywane przez klasę Rys. Klasa Rys dziedziczy funkcje z klasy Ssaki, która z kolei dziedziczy po klasie zwierzęta. Innymi słowy, kiedy tworzymy obiekt Rys, możemy użyć funkcji zdefiniowanych w klasie Rys, a także tych funkcji, które zostały zdefiniowane w klasach ssaków i zwierząt.
Na tej samej zasadzie, jeśli stworzymy obiekt klasy Ssak, możemy użyć funkcji zdefiniowanych w tej klasie, a także tych z klasy nadrzędnej: Zwierzeta.



Wywoływanie funkcji z funkcji

Kiedy wywołujemy funkcje na obiekcie, używamy zmiennej obiektu. Jeśli chcemy wywołać funkcję wewnątrz jeden z funkcji obiektów np. przed jedzeniem chcemy się poruszyć to robimy to wykorzystując parametr **self**. Parametr **self** jest sposobem, aby jedna funkcja w klasie mogła wywołać inną. Załóżmy, że dodaliśmy nową funkcję o nazwie *znajdzJedenie* do klasy Rys:

```
class Rys(Zwierzeta):
    def znajdzJedzenie(self):
        self.Ruch()
        print("Mam jedzenie!")
        self.Jedzenie()
```

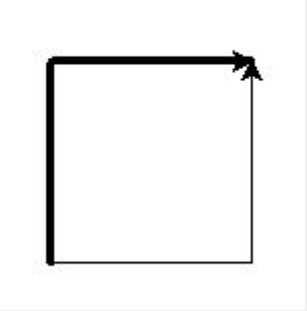
Obiekty i klasy na rysunkach

Co powiesz na bardziej graficzne podejście do obiektów i klas?
Wróćmy do naszego żółwika i modułu *turtle.* Kiedy używaliśmy `turtle.Pen()`, Python tworzył obiekt klasy `Pen` zawartej w module *turtle* (podobnie jak nasze obiekty *Rysio* i *Eustachy* wcześniej w notatce). Możemy stworzyć dwa żółwiki, tak jak tworzyliśmy dwa rysie:

```
import turtle
leon = turtle.Pen()
honorata = turtle.Pen()
```

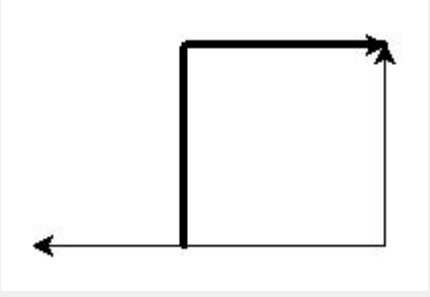
Każdy żółw (*leon* i *honorata*) jest członkiem klasy `Pen`. W tym momencie obiekty pokazują swoją prawdziwą moc. Mając stworzone obiekty żółwi, możemy używać funkcji na każdym z nich, przez co żółwiki będą rysować niezależnie od siebie (co nie oznacza równocześnie - niestety).
Poniższy kod pokazuje, jak możemy stworzyć rysunek przedstawiony obok:

```
leon = turtle.Pen()
honorata = turtle.Pen()
leon.pensize(4)
honorata.forward(100)
honorata.left(90)
honorata.forward(100)
leon.left(90)
leon.forward(100)
leon.right(90)
leon.forward(100)
```



Mamy teraz dwa żółwie skierowane w dwóch różnych kierunkach - leon zmierza w prawo, a honorata w górę.
Stwórzmy trzeciego żółwia (którego nazwiemy norbert) i przejdźmy nim trochę w lewo, dopisując poniższy kod:

```
norbert = turtle.Pen()
norbert.left(180)
norbert.forward(75)
```



Zapamiętaj, że za każdym razem gdy wywołujemy `turtle.Pen()`, dodajemy nowy, niezależny obiekt. Każdy obiekt jest instancją(wikipedia) klasy `Pen` i na każdym możemy użyć tych samych funkcji, ale przez to, że używamy obiektów, możemy poruszać każdym żółwiem niezależnie. Tak jak nasze niezależne obiekty rysiów (Rysio i Eustachy) Leon, Honorata i Norbert są niezależnymi obiektami żółwi.
Jeśli stworzymy nowy obiekt o tej samej nazwie co wcześniej utworzony, stary obiekt niekoniecznie zniknie. Spróbuj sam: utwórz kolejnego żółwia leon i spróbuj się nim przemieszczać.

Inicjowanie obiektu

Czasami podczas tworzenia obiektu, chcemy by ten miał już ustawione pewne właściwości do przyszłego użycia.
Mówiąc o *inicjalizacji* obiektu, mamy na myśli przygotowanie go, do późniejszego użycia.

Załóżmy na przykład, że chcemy ustawić liczbę cętków na naszych obiektach rysi (tak, rysie mają cętki), gdy zostaną utworzone - to znaczy, gdy zostaną zainicjowane. By to zrobić, tworzymy funkcję `__init__` (zauważ, że po obu stronach słowa piszemy dwa znaki podłogi - '_').

`__init__` to specjalny typ funkcji w klasach Pythona i musi on mieć dokładnie taką nazwę. Funkcja *init* to sposób na ustawienie właściwości dla obiektu, gdy jest tworzony. Python automatycznie wywoła tę funkcję, gdy stworzymy nowy obiekt.
Przykład:

```
class Rysie:
    def __init__(self, cetki): #L1
        self.cetki_rysia = cetki #L2
```

L1 - definiujemy funkcję *init* z dwoma parametrami - *self* i *cetki*
Zupełnie jak inne funkcje, które definiujemy w klasie, funkcja *init* również musi posiadać parametr *self* (jako pierwszy).

L2 - do zmiennej obiektowej (właściwości obiektu) *cetki_rysia* przypisujemy zmienną *cetki* za pomocą parametru self. Możesz myśleć o tej linii kodu w ten sposób:
"Weź wartość parametru *cetki* i zapisz go na później (używając do tego zmiennej obiektowej *cetki_rysia*)."

Tak samo jak jedna funkcja w klasie może wywoływać inne funkcje za pomocą parametru *self,* zmienne w klasie są również dostępne za pomocą *self*.

Następnie, jeśli stworzymy kilka nowych obiektów rysi (np. Arnold i Henryk) i wyświetlimy ich ilość cętków, możemy zobaczyć funkcję inicjalizującą w akcji:

```
arnold = Rysie(100) #L3
henryk = Rysie(39)
print(arnold.cetki_rysia) # zostanie wypisane 100
print(henryk.cetki_rysia) # zostanie wypisane 39
```

L3 - tworzymy instancję klasy Rysie, używając wartości parametru 100. Wywołuje to w ten sposób funkcję *__init__* i używa 100 jako wartość parametru *cetki*. W następnej linii kodu robimy to samo z wartością 39.

Zapamiętaj - gdy tworzymy obiekt klasy, tak jak Arnold, możemy odwoływać się do jego zmiennych lub funkcji używając operatora kropki i nazwy zmiennej/funkcji, której chcemy użyć (np. `arnold.cetki_rysia`).
Lecz jeśli tworzymy funkcję wewnątrz klasy, odwołujemy się do tych samych zmiennych (oraz innych funkcji) za pomocą parametru *self* (np. `self.cetki_rysia`).

Podsumowanie

Używaliśmy klas, by stworzyć kategorię “cośków” i tworzyliśmy obiekty (instancje) tych klas.

Nauczyliśmy się jak dziecko klasy dziedziczy funkcje swojego rodzica i wiemy, że nawet gdy dwa obiekty są tej samej klasy, niekoniecznie są swoimi klonami. Na przykład - obiekt ryś może mieć własną ilość cętków.

Nauczyliśmy się też jak wywoływać (lub uruchamiać) funkcje na obiektach i tego, jak zmienne obiektu są sposobem zapisywania wartości w tych obiektach.

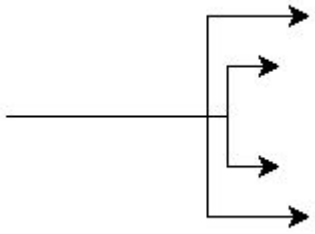
Używaliśmy parametru *self* w funkcji, by odwołać się do innych funkcji i zmiennych.

Pojęcia zawarte w tej notatce są fundamentalną wiedzą dotyczącą Pythona, której z pewnością będziemy używać w przyszłych notatkach (i kolejnych zajęciach).

Zagadki programistyczne

#1 Żółwie widły

Stwórz program, który narysuje widły:



#2 Król stada rysiów

Stwórz klasę Rysie, w której będziesz mógł zainicjować wiek rysia. Potem stwórz 3 instancje tej klasy, niech każdy ryś będzie w innym wieku. Następnie stwórz funkcję `get_biggest_number()`, pobierającą dowolną ilość parametrów wieku (*args) i zwracającą największy z nich. Na koniec wypisz wiek naszego króla stada jak na przykładzie:

```
Najstarszy ryś ma 10 lat.
```

#3 Taneczne rysie

Stwórz klasę Rysie, w której napiszesz funkcje do poruszania lewą i prawą nogą do przodu i do tyłu.
Funkcja do poruszenia lewą nogą do przodu może wyglądać np. tak:

```
def lewaNogaPrzod(self):
    print('lewa noga w przód!')
```

Potem stwórz funkcję *taniec*, by nauczyć Eustachego tańczyć (funkcja wywoła wszystkie funkcje poruszania nogami które stworzyłeś).
Przykładowy wynik zadania:

```
Eustachy = Rysie()
Eustachy.taniec()
#W konsoli zostanie wypisane:
#lewa w przód!
#lewa w tył!
#prawa w przód!
#prawa w tył!
#lewa w tył!
#i obróóót! (ok, to nie jest wymagane, poniosło mnie)
```